MASTER EN SCIENCES INFORMATIQUES

Analyse et implémentation d'un environnement de rangement d'informations

Robert, Stefan

Award date:
1986

Awarding institution:
Universite de Namur

Facultés Universitaires Notre-Dame de la Paix à Namur

Institut d'informatique

Analyse et implémentation d'un environnement de rangement d'informations.

Mémoire présenté par
Stefan ROBERT
en vue de l'obtention du titre de
Licencié et Maître en Informatique

Promoteur : R. Lesuisse

Année académique 1985-86.

Qu'il nous soit permis de remercier les différentes personnes qui ont collaboré d'une manière ou d'une autre à l'élaboration de ce mémoire :

Tout d'abord, M. Lesuisse, promoteur du présent mémoire, qui a apporté son aide assidue et chaleureuse tout au long de ce travail.

La société BIM, de Everberg, ainsi que Bernard Geubelle et les assistants de l'Institut, Philippe Delhaye et Thérèse Collet-PetitJean, sans lesquels il eût été impossible de travailler sur une station de travail SUN-2.

Les chercheurs Carine Charlot et Benoît Vanhoutte pour leurs conseils précieux sur les Systèmes de Gestion de Bases de Données (SGBD).

M. van Lamsweerde et Sophie Leonard pour leurs indications judicieuses sur les réseaux sémantiques.

M. Barreto pour son apport de connaissances des langages orientés objet et plus particulièrement Methods, une version de Smalltalk sur PC.

Jean-Marie Jacquet pour ses conseils sur les langages Lisp et Prolog.

La société C.I.G., de Bruxelles, et plus particulièrement Bruno Delcourt de l'Institut pour l'utilisation du VAX-750 ainsi que son aide dans la pratique du Unix, du langage C et du SGBD relationnel Ingres.

Qu'il nous soit permis, enfin, de mettre à profit ce travail de fin d'études pour remercier vivement les professeurs de l'Institut d'Informatique qui durant trois années nous ont, à travers leurs cours, permis de réunir les matériaux nécessaires à la bonne conduite de ce mémoire.

PLAN DU MEMOIRE

INTRODUCTION

CHAPITRE 1 : DESCRIPTION D'UN ENVIRONNEMENT DE RANGEMENT D'INFORMATIONS : LE BUREAU

1. INTRODUCTION 1

2. L'INFORMATION 2

3. LES MEUBLES DE RANGEMENT D'INFORMATIONS 3

3.1. L'ARMOIRE
3.2. L'ETAGERE 5
3.3. LA BOITE 6
3.4. LA POUBELLE 7

4. LES UTILISATEURS 8

5. LES MANIPULATIONS D'INFORMATIONS 9

5.1. La méthode d'accès aux informations 9
5.2. L'accès aux informations 10
5.3. L'ajoût d'une information 11
5.4. La consultation d'une information 12

6. CONCLUSION 13

CHAPITRE 2 : DESCRIPTION D'UN ENVIRONNEMENT DE RANGEMENT AUTOMATISE D'INFORMATIONS

0. PREAMBULE 1

1. INTRODUCTION

2. LES CONCEPTS DE BASE DE L'E.R. D'INFORMATIONS 2

2.1. LES UTILISATEURS ET LES GROUPES D'UTILISATEURS

2.1.1. L'UTILISATEUR 3
2.1.2. LE GROUPE D'UTILISATEURS
2.1.3. LA METHODE D'ACCES AUX INFORMATIONS
2.1.4. L'ORGANISATION DES GROUPES D'UTILISATEURS 5

2.2. LES TYPES D'OBJET PHYSIQUES 6

2.3. LES TYPES D'OBJET LOGIQUES

2.3.1. LA STRUCTURE GENERALE DE L'INFORMATION

2.3.1.1. L'INFORMATION
2.3.1.2. LA STRUCTURE DE L'INFORMATION 7

2.3.2.3. ORGANISATION DE L'INFORMATION 12

3. LES OPERATIONS SUR LES OBJETS DE L'E.R. 13

4. CONCLUSION 15

CHAPITRE 3 : ANALYSE FONCTIONNELLE DE L'ENVIRONNEMENT DE RANGEMENT DINFORMATIONS

1. INTRODUCTION 1

2. SPECIFICATION DES TYPES D'OBJET DE L'ENVIRONNEMENT DE RANGEMENT

2.1. Le schéma Entité-Association (E-A) de 1'E.R.

2.2. Interprétation du schéma E-A 3

2.2.1. Les types d'entité (T.E)

2.2.2. Les attributs des T.E. 4

2.2.3. Les types d'associations (T.A.) 28

2.2.4. Les contraintes d'intégrité (C.I.) du schéma E-A 33

2.2.4.1. Les C.I. sur les entités

2.2.4.2. Les C.I. sur les associations

2.2.4.3. Les C.I. sur les attributs 39

2.3. Autres données 40

3. SPECIFICATION DES OPERATIONS SUR LES OBJETS DE L'E.R. 41

3.1. PORTEE DES OPERATIONS

3.2. LES OPERATIONS 43

CHAPITRE 4 : IMPLEMENTATION

1. INTRODUCTION 1

2. DESCRIPTION DE LA CONFIGURATION MATERIELLE ET LOGICIELLE

3. REPRESENTATION DES DONNEES 2

3.1. Analyse fonctionnelle
3.2. Conception logique 4
3.3. Conception physique 8

4. ARCHITECTURE DE L'APPLICATION 10

4.1. Architecture logique
4.2. Architecture physique 11

5. CONCLUSION 12

CHAPITRE 5 : EVALUATION

1. INTRODUCTION 1

2. MODELE D'E.R. D'INFORMATIONS

3. MODELE CONCEPTUEL DE REPRESENTATION DES DONNEES 3

4. OUTILS D'IMPLEMENTATION 6

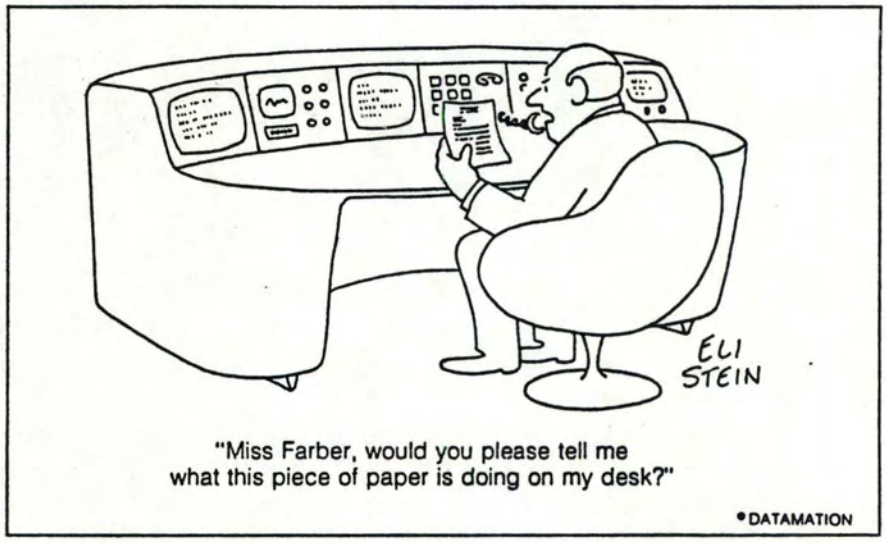
5. PERFORMANCES DE L'IMPLEMENTATION 11

6. PERSPECTIVES POUR LE MEMOIRE 13

7. CONCLUSION 14

CONCLUSION

La Bureautique De Demain . . .



When will there be a truly paperless office ?

INTRODUCTION

"Les systèmes d'information de demain devront pouvoir saisir, stocker, retrouver, manipuler et présenter des informations de toutes espèces. Ils exigeront des interfaces conviviaux , de tester possibilités de calcul et de raisonnement, et des accès partagés à de grandes banques d'informations. Si la technologie hardware demandée commence à apparaître acceptable, la technologie software correspondante pour construire des systèmes ne l'est pas encore. Des améliorations notables dans la productivité software seront rendues possibles par des environnements avancés de développement d'application basés sur de puissantes technologies et de nouveaux langages."

"Les concepts, techniques et outils nécessaires pour la conception, l'implémentation et l'utilisation des systèmes d'information futurs seront le résultat de l'intégration de ceux développés et utilisés actuellement dans des domaines séparés de la Science Informatique. Un domaine clé est l'Intelligence Artificielle (IA), qui fournit des bases de connaissances reposant sur des théories sémantiques de l'information en vue d' une interprétation correcte. Un domaine aux potentialités équivalentes est représenté par les Bases de Données (BD) qui offrent les moyens de construire et de maintenir de grandes BD partagées, basées sur des théories de calcul de l'information en vue d'un traitement efficace. Les Langages de Programmation constituent un troisième domaine important. Ils proposent des outils performents pour la mise en oeuvre de grands programmes. Pour suivre l'évolution des besoins, viennent s'ajouter les réseaux informatiques, les architectures de machine et la bureautique." (TSIC85)

Ce processus d'intégration commence à se réaliser dans le concept de Station de Travail qui paraît être l'outil bureautique le plus avancé à l'heure actuelle. Il rassemble une interface conviviale (écran graphique bit-map, icones, multi-fenêtre, multi-tâche, souris, ...), des facilités de communication entre stations par interconnection sur un réseau local type Ethernet, par exemple, et un partage des ressources : disque dur à grande capacité, jukebox de disques optiques, imprimante laser, scanner, ... Les stations les plus connues sont Star, Sun, Appolo, Filenet, Tektronix et à plus petite envergure Lisa et Macintosh. (XEROX85) (SUN85) (OLDO85) (BYTE84) (WEBS84)

INTRODUCTION

Dans le domaine qui nous intéresse pour ce mémoire, la bureautique, certaines réalisations individuelles ont été réalisées, et continuent à être, au sein de l'Institut d'Informatique, dans le cadre de séminaires (LESU85) ou de mémoires. Citons par exemples :

- l'étude et la réalisation d'un courrier électronique sur un réseau local (BEJO85) ;

- une gestion de courrier électronique inter-réseaux (GAVA86) ;

- l'étude d'une messagerie vocale (GOHE85) ;

- l'étude du concept de document dans un langage de spécification sur l'atelier logiciel IDA (DACH86) ;

- une gestion de fenêtres sur une station de travail SUN-2 (LOPI86) ... ;

Dans le cadre de ce mémoire, nous nous intéresserons plus particulièrement au problème du rangement, classement des informations dans un système automatisé et ceci en vue de la gestion (ajout, modification, suppression) et la recherche de celles-ci.

Dans un premier chapitre, nous esssayerons de décrire un environnement de rangement manuel d'informations : le bureau.

Dans un deuxième chapitre, nous nous baserons sur cette description pour modéliser un environnement de rangement automatisé.

Le troisième et quatrième chapitre seront consacrés à l'analyse fonctionnelle et à l'implémentation de cet E.R. automatisé.

Un cinquième chapitre dressera une évaluation complète du travail en essayant de proposer des améliorations et quelques idées en vue de la continuation de ce mémoire.

CHAPITRE 1 : DESCRIPTION D'UN ENVIRONNEMENT DE RANGEMENT D'INFORMATIONS : LE BUREAU

1. INTRODUCTION

Dans ce premier chapitre, nous tenterons d'approcher de manière intuitive ce que nous entendons par environnement de rangement (E.R.).

Pour cela, nous nous référerons à la manière dont les informations sont actuellement rangées dans un environnement classique : le bureau.

Nous entendons par bureau, la pièce qui sert de lieu de travail à une ou plusieurs personnes et, en particulier, de lieu de rangement d'informations. Ces personnes pourront être des secrétaires, des employés, des cadres, ...

Nous décrirons successivement les informations, les meubles permettant leur rangement, les utilisateurs et les manipulations de ces informations.

2. L'INFORMATION

L'information est l'objet de base manipulé dans un bureau. Il en existe différents modes de représentation. Citons les modes : oral, écrit, graphique ou mixte (figure 1-1).

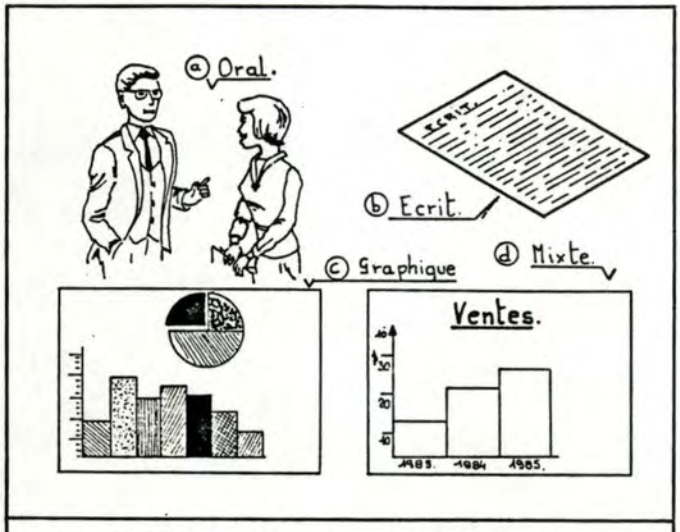


Fig. 1-1 : Modes de représentation de l'information

Une liste non exhaustive d'informations serait : une lettre d'un fournisseur, un mémo, un formulaire d'inscription, une facture, le compte rendu d'un appel téléphonique, les résultats graphiques d'un exercice comptable, ...

Ces informations, lorsqu'elles ne sont pas utilisées par une personne, sont rangées (classées) dans des meubles de rangement.

3. LES MEUBLES DE RANGEMENT D'INFORMATIONS

Dans un bureau, il existe une variété de meubles pour le rangement d'informations. Nous reprendrons simplement l'armoire, l'étagère, la boîte et la poubelle.

3.1. L'armoire

L'armoire est un meuble plus ou moins volumineux, le plus souvent composé de divers compartiments, les tiroirs.

Ces tiroirs peuvent abriter des informations ou des fardes contenant elles-mêmes des informations. Nous voyons déjà que dans ce cas nous avons à faire à une structure de classement à plusieurs niveaux comme nous le montre la figure 1-2.

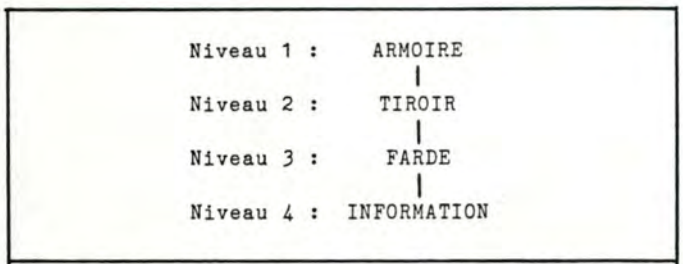


Fig. 1-2 : Niveaux de structuration d'une armoire

Une étiquette peut être collée sur chaque tiroir indiquant les informations qu'il contient. Par exemples :

. une année, pour un classement chronologique ;

. un intervalle de lettres, pour un classement alphabétique.

La figure 1-3 illustre une représentation possible de l'armoire.

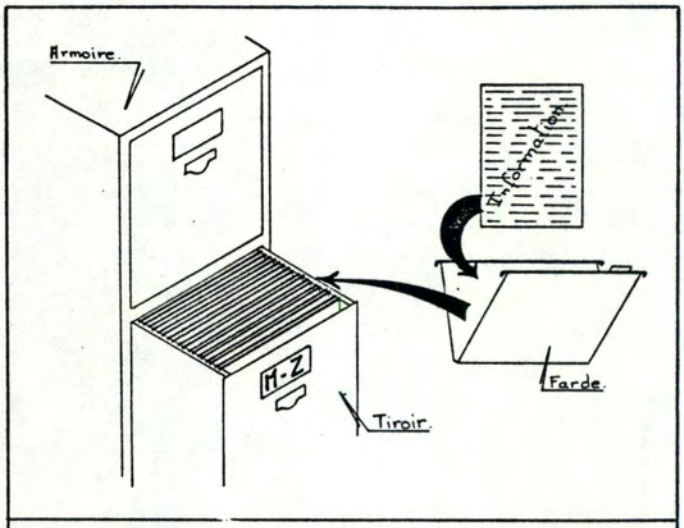


Fig. 1-3 : Représentation d'une ARMOIRE

3.2. L'étagère

Tout comme l'armoire, l'étagère est composée de différents compartiments représentables sous forme de travées et de colonnes.

Chaque compartiment peut servir de support à des boîtes, des fardes ou directement des informations, par exemple des ouvrages.

La nécessité d'indiquer ce que chaque compartiment de l'étagère contient n'est pas aussi importante que pour l'armoire dans la mesure où la plupart des étagères sont ouvertes.

La figure 1-4 illustre la représentation qui sera utilisée pour l'étagère.

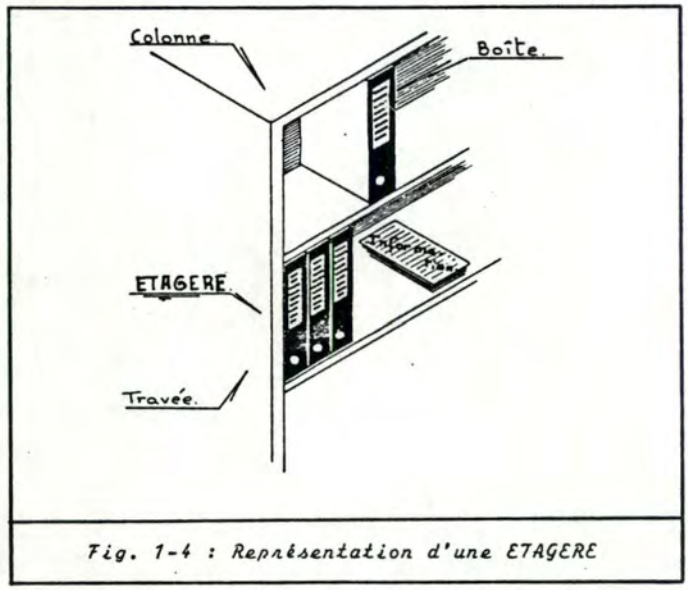


Fig. 1-4 : Représentation d'une ETAGERE

3.3. La boîte

La boîte est un moyen de rangement alternatif, souvent moins volumineux que l'armoire. Elle sera, par exemple, une boîte de carton du format d'un grand classeur que l'on rangera sur une étagère de manière modulable (figure 1-4).

Elle peut jouer un rôle spécifique tel que celui de rangement du courrier. Une boîte appelée "IN" contient le courrier reçu et une boîte appelée "OUT" contient le courrier à expédier. Une personne chargée de la poste interne vient périodiquement apporter le courrier destiné au bureau et relever le courrier à destination de l'extérieur ou d'un autre bureau.

La boîte contient souvent une indication reprenant le type de contenu ou son rôle : par exemples : COPIES EXAMENS, IN, OUT.

La figure 1-5 reprend les types de boîtes décrites ci-dessus.

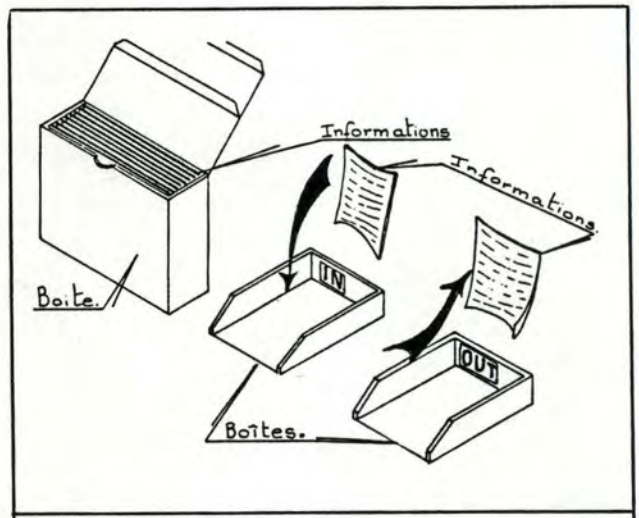


Fig. 1-5 : Représentation de BOITES

3.4. La poubelle

La poubelle peut également être considérée comme un meuble d'informations dont la particularité est d'être considérées comme inutiles.

Il arrive qu'une information ait été mise dans la poubelle par mégarde. Il est cependant possible de la retirer de la poubelle si l'on s'y prend à temps. Un personnel d'entretien vient régulièrement vider la poubelle de son contenu pour éviter qu'elle ne déborde et pour donner une image de propreté au bureau.

La figure 1-6 donne une représentation d'un type de poubelle.

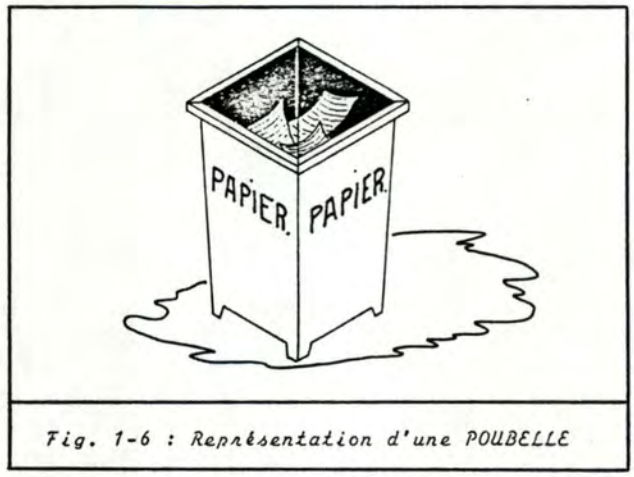


Fig. 1-6 : Représentation d'une POUBELLE

4. LES UTILISATEURS

Les utilisateurs sont l'ensemble des personnes qui ont besoin d'accéder au bureau pour y apporter, consulter ou retirer des informations.

Nous considérons donc que le bureau est une pièce qui peut contenir des informations partagées entre plusieurs utilisateurs.

On constatera souvent que dans un bureau, une personne est désignée responsable de la mise en ordre des informations. Cette personne sera particulièrement bien informée de l'organisation du bureau.

5. LES MANIPULATIONS D'INFORMATIONS

Trois types de manipulations seront présentées : l'accès aux informations, l'ajout de nouvelles informations et leur consultation.

5.1. La méthode d'accès aux informations

Selon le type d'information, l'accès aux informations peut se faire soit directement par l'utilisateur, soit par l'intermédiaire du responsable du bureau.

S'il s'agit d'une information directement accessible à tous, rien n'empêche un utilisateur d'en prendre connaissance. C'est par exemple le cas de certaines revues mensuelles auxquelles le bureau est abonné.

Si par contre, l'information n'est accessible qu'à un groupe restreint de personnes, tout utilisateur doit passer par le responsable qui vérifiera s'il appartient bien à ce groupe avant de lui donner l'accès à l'information. Il s'agira, par exemple, d' informations financières seulement accessibles au personnel de la comptabilité.

Enfin, pour les informations confidentielles de chaque utilisateur, une facilité devrait pouvoir leur être donnée, par exemple par l'utilisation d'une clé qui leur sera propre et qui leur donnera accès à l'armoire contenant l'information. On peut éventuellement prévoir un double de chaque clé pour le responsable du bureau, qui serait utilisée quand l'utilisateur oublierait la sienne. Ce serait par exemple le cas de certains livres de compte qui doivent être mis en lieu sûr chaque soir.

On s'aperçoit peu à peu que le responsable du bureau doit être une personne respectée et digne de confiance pour remplir la tâche qui lui est assignée.

5.2. L'accès aux informations

Pour accéder à une information du bureau, il faut avoir une idée de sa localisation ou de sa description (figure 1-7).

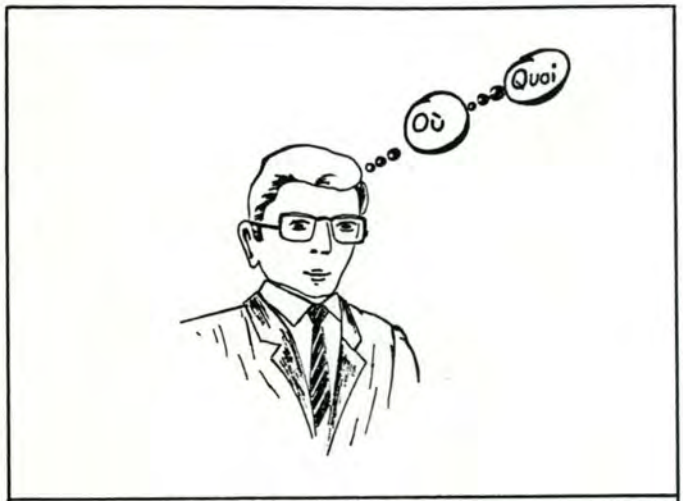


Fig. 1-7 : Modes d'accès aux informations

La connaissance de la localisation est le moyen le plus direct pour accéder aux informations. Par exemple : on sait que l'information se trouve dans une farde portant la mention "FACTURES 1986" dans l'armoire réservée à la comptabilité.

A défaut de la localisation, il faut s'adresser au responsable du bureau en lui donnant la description la plus précise possible de l'information recherchée. Connaissant l'organisation du bureau, celui-ci pourra peut-être donner une indication sur la localisation de l'information.

Si le nombre d'informations contenues dans le bureau devient trop important, la mémoire du responsable risque de ne plus suffire plus pour contrôler toutes les informations du bureau.

Un autre moyen de gérer les informations consiste à faire appel à un répertoire sous forme d'un livre ou d'un système à fiches. Par exemple un système à fiches comme ceux utilisés dans certaines bibliothèques où à partir du nom d'un auteur on peut retrouver tous ses ouvrages acquis par la bibliothèque accompagnés de leur localisation. Par exemple I 391/13/1, I correspondant au domaine INFORMATIQUE, 391 à la rubrique BUREAUTIQUE, 13 au numéro d'ouvrage dans cette rubrique et 1 indiquant qu'il s'agit du premier exemplaire (figure 1-8).

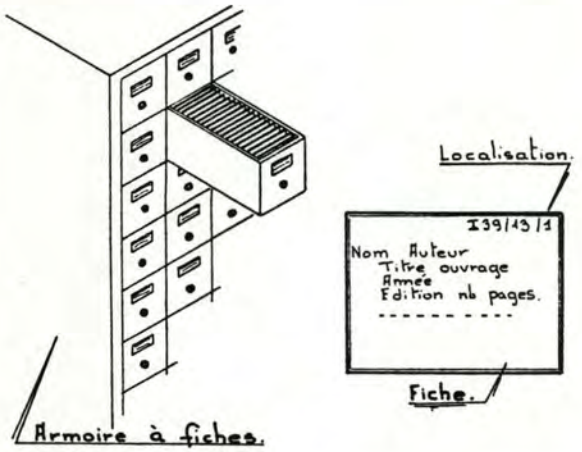


Fig. 1-8 : Représentation d'un système à fiches

Un tel système demande un travail important de mise à jour des fiches, correspondant à l'évolution de l'ensemble des informations ; toute information nouvelle, ici ouvrage, exige la création d'une nouvelle fiche.

Certains systèmes à fiches permettent d'autres types de recherche. Par exemple, outre le classement par auteur, les fiches peuvent être classées par thème, chacune d'elles reprenant alors les ouvrages correspondant au thème. Dans ce cas, il s'agit d'un travail supplémentaire qui peut être fait indépendamment du premier classement. On voit cependant qu'un tel système manuel est limité par le temps nécessaire à la mise à jour des fiches. Il ne peut de plus être envisagé que pour un volume limité d'informations.

5.3. L'ajoût d'une information

L'ajout d'une information passe de préférence, sauf pour les informations confidentielles, par le responsable du bureau. Il est en effet la personne qui connaît le mieux l'organisation des dernières. Il pourra donc juger de l'endroit où il devra insérer cette information de manière à pouvoir la retrouver ultérieurement.

S'il s'agit d'une information confidentielle, seul son propriétaire a besoin de connaître sa localisation car il sera le seul à pouvoir y accéder ultérieurement.

5.4. La consultation d'une information

La consultation d'une information est précédée de son accès suivant la procédure décrite ci-dessus et de son retrait temporaire de son meuble de rangement.

Ce retrait sera le plus souvent signalé par une indication à l'endroit de son rangement, comportant au minimum le nom de l'utilisateur qui a "emprunté" l'information, éventuellement complété par la date d'emprunt, une estimation du délai maximum d'emprunt (date restitution), ...

Cette indication a une double utilité :

- d'une part, elle signale l'emprunt de l'information. Sa présence représente la seule trace de l'existence de cette information dans le bureau ;

- d'autre part, elle permet en cas de besoin de retrouver l'information par l'identité de la personne qui l'a "empruntée".

6. CONCLUSION

Ce chapitre a permis de dresser la description d'un environnement de rangement classique et manuel d'informations : le bureau.

Nous y avons décrit les informations, les meubles permettant leur rangement, les utilisateurs et les différentes manipulations de ces informations.

Dans le chapitre suivant nous nous baserons sur cette description pour concevoir un modèle d'environnement de rangement d'informations automatisé.

CHAPITRE 2 : DESCRIPTION D'UN ENVIRONNEMENT DE RANGEMENT D'INFORMATIONS AUTOMATISE

0. PREAMBULE

Nous avons jugé préférable de développer l'analyse fonctionnelle en deux chapitres plutôt qu'en un seul ; ceci pour faciliter la compréhension de notre application.

Dans un premier chapitre (chapitre 2), nous décrirons un E.R. d'informations automatisé, par comparaison avec la description de l'E.R. manuel décrit au premier chapitre. Nous n'insisterons pas encore sur la forme définitive de cet environnement. Ce chapitre permettra de préciser quelques concepts utilisés et d'introduire, de manière intuitive, le lecteur dans un E.R. automatisé.

Dans un deuxième chapitre (chapitre 3) sera présentée une description plus formelle de l'E.R. sous forme du modèle Entité-Association (E-A) pour les données et de modules pour les opérations. C'est cet E.R. que nous nous proposerons d'implémenter par la suite.

1. INTRODUCTION

Après avoir décrit, dans le premier chapitre, la composition et l'utilisation d'un E.R. classique de bureau, nous essayerons d'analyser ce que serait un E.R. automatisé.

L'analyse effectuée dans le premier chapitre servira de base et de comparaison pour notre E.R. automatisé.

Nous essayerons de ne pas perdre de vue que l'objectif principal de notre E.R. d'informations sera de faciliter le classement, la manipulation et la recherche d'informations (1).

Nous verrons successivement les éléments qui font partie de notre environnement. Ensuite nous verrons les opérations applicables sur les éléments de celui-ci.

---

(1) Le terme INFORMATION est repris de (LESU85). Il sera clairement spécifié au point 2.3.1.

2. LES CONCEPTS DE BASE DE L'E.R. D'INFORMATIONS

Ce qui caractérise un E.R. dans sa fonction de rangement, ce sont ses meubles. Nous allons donc meubler notre E.R. d'ARMOIRES constituées de TIROIRS, de BOITES, de FARDES et d'une POUBELLE. Les objets rangés sont des informations et des groupes d'informations (2).

Ces différents éléments seront appelés les TYPES D'OBJET de l'E.R. Nous dirons aussi qu'ils constituent deux classes de types. La première classe reprend les types d'objet dits PHYSIQUES. Ce sont les armoires, les tiroirs, les boîtes, les fardes et la poubelle. Ils correspondent à des structures physiques de rangement. La deuxième classe reprend les types d'objet dits LOGIQUES : ce sont les informations et les groupes d'informations, les types d'objet classés.

Avant d'entrer dans la description de ces éléments, introduisons un autre concept, aussi important que les types d'objet permettant le rangement des informations : le concept d'utilisateur, acteur de l'E.R.

2.1. LES UTILISATEURS ET LES GROUPES D'UTILISATEURS

Dans l'E.R., un certain nombre de personnes, appellées utilisateurs, voudront accéder aux informations. Elles utiliseront, à cet effet, un mot de passe. Pour que l'utilisation de l'E.R. soit acceptable, ces personnes s'attendent à ce que certaines de leurs exigences soient rencontrées. Ainsi :

- un premier souhait serait que cet E.R. puisse servir de lieu de rangement unique de toutes leurs informations quelle que soit leur nature ;
- ils voudraient pouvoir y ranger :
  - des informations qui puissent être consultées par tout le monde ;
  - d'autres qui ne pourraient être accessibles qu'à un groupe particulier de personnes ;
  - et enfin, des informations confidentielles qui ne pourraient être accessibles qu'à leur propriétaire.
- de même que dans toute organisation humaine, les utilisateurs voudraient que certains d'entre eux soient habilités à accomplir certaines tâches (par ex. : ajouter une armoire dans le bureau) qui soient refusées à d'autres ;

---

(2) Le terme GROUPE D'INFORMATIONS correspond au concept fichier/dossier/pile dans (LESU85). Il sera clairement spécifié au point 2.3.2.

- d'autre part, les personnes aimeraient pouvoir retirer une information de l'E.R. en vue de la consulter ou de la mettre à jour avant de la remettre en place tout en signalant ce retrait aux autres utilisateurs qui voudraient manipuler la même information au même moment.

Ces exigences font apparaître la nécessité d'une certaine discipline dans l'utilisation de l'E.R. si l'on veut éviter des problèmes tels que la disparition d'informations ou l'indiscrétion de certains utilisateurs.

Dans les points suivants seront définis les concepts d'utilisateur et de groupe d'utilisateurs. On y verra aussi comment organiser les groupes d'utilisateurs en fonction de la manière dont les informations sont accédées.

2.1.1. L'UTILISATEUR

Un utilisateur est une personne, individu qui peut manipuler les informations de l'E.R. Tout accès à l'E.R., pour être valide, devra être accompagné de l'identité de l'utilisateur, reconnu comme tel par l'E.R.

A la création de l'E.R., il y a automatiquement création du premier utilisateur particulier : l'administrateur de l'E.R. Il aura, par rapport aux autres utilisateurs, des droits particuliers qui seront indiqués progressivement. Signalons dès à présent qu'il n'y a que l'administrateur qui pourra ajouter de nouveaux utilisateurs.

2.1.2. LE GROUPE D'UTILISATEURS

Un groupe d'utilisateurs rassemble tous les utilisateurs qui doivent, de par leur fonction dans un bureau, accéder aux mêmes informations. Ce regroupement par fonctionalité permet d'alléger la procédure de contrôle de l'accès aux informations (WOO83).

2.1.3. LA METHODE D'ACCES AUX INFORMATIONS

Une méthode d'accès possible aux informations serait de fixer, pour chaque utilisateur, la liste des informations auxquelles il peut accéder. Cette méthode présente cependant plusieurs inconvénients. En l'occurrence, la suppression d'une information exige la mise à jour des listes d'accès (L.A.) de tous les utilisateurs qui ont accès à cette information. De même, la création d'une nouvelle information demande la mise à jour des L.A. de tous les utilisateurs qui auront accès à cette nouvelle information.

Une méthode alternative serait de fixer, pour chaque information, la liste des utilisateurs qui y ont accès. Cette

méthode n'est pas encore parfaite à cause de la similarité de la L.A. de certaines informations. Cette similarité des L.A. est liée à la nature du travail effectué par les utilisateurs dans un bureau. De plus, un utilisateur peut changer de responsabilité dans un bureau, suite à une promotion par exemple. Un tel changement nécessite la mise à jour de la L.A. des informations auxquelles il pourra et ne pourra plus accéder à l'avenir.

Nous regrouperons les utilisateurs en groupes d'après la fonction qu'ils exercent dans le bureau. Ces groupes représenteront donc l'organisation humaine du bureau. Cette méthode présente plusieurs avantages.

Tout d'abord, elle allège la gestion des L.A. des informations ; seuls les groupes d'utilisateurs y sont repris. De plus, les groupes représentant l'organisation, sont moins sensibles aux variations des utilisateurs. Finalement, un utilisateur peut changer de groupe sans que l'on ait à mettre à jour les L.A. (WOO83).

Outre ces L.A. d'autres renseignements permettent de contrôler l'accès aux informations :

- le nom de l'utilisateur ayant créé l'information ainsi que la date de création ;

- le nom de l'utilisateur ayant consulté pour la dernière fois l'information ainsi que la date de cette consultation ;

- le nom de l'utilisateur qui a mis à jour pour la dernière fois l'information accompagné de la date de cette mise à jour.

Enfin, le propriétaire de l'information (l'utilisateur qui l'a créée) a la possibilité de la rendre confidentielle c'est-à-dire que lui seul pourra encore y accéder.

La méthode d'accès à l'information qui vient d'être décrite sera partiellement généralisée à tous les types d'objet de l'E.R.

2.1.4. L'ORGANISATION DES GROUPES D'UTILISATEURS

Pour poursuivre la comparaison avec l'organisation humaine du bureau, il est possible d'établir une relation hiérarchique entre les différents groupes d'utilisateurs pour l'accès aux informations.

Cette relation peut être représentée sous la forme d'une arborescence (figure 2-1).

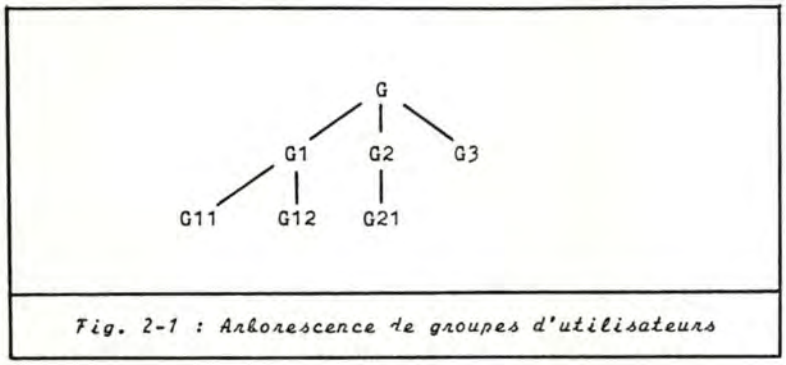


Fig. 2-1 : Arborescence de groupes d'utilisateurs

Comme dans une organisation humaine où les pouvoirs dépendent du niveau hiérarchique auquel on se trouve, l'accès aux informations est organisé sur base de la place dans la hiérarchie.

Ainsi une information accessible au groupe G11 est automatiquement accessible, en plus de leurs possibilités d'accès propres, à tous les groupes parents en lignée directe, c'est-à-dire les groupes G1 et G.

L'inverse n'est pas vrai. Une information accessible au groupe G1 n'est pas nécessairement accessible au groupe G11 sauf si ce droit est mentionné.

Il est clair que, dans cette organisation, la place de l'administrateur de l'E.R. est au sommet de la hiérarchie, c'est-à-dire le groupe G. En effet, il a besoin d'avoir accès à toutes les informations pour pouvoir remplir son rôle de responsable.

Pour que la méthode d'accès aux informations soit efficace, les utilisateurs devront faire partie des bons groupes et ces groupes devront refléter le modèle de l'organisation humaine du bureau.

On peut remarquer la ressemblance de cette organisation des groupes d'utilisateurs avec l'organisation des directoires sur le système d'exploitation UNIX (BOUR) (SOBE84).

2.2. LES TYPES D'OBJET PHYSIQUES

Rappelons que les types d'objet physiques retenus pour l'E.R. sont les mêmes que ceux décrits au premier chapitre pour le bureau à un type d'objet près : l'étagère. Elle n'a pas été reprise parce que son rôle se confond avec celui de l'armoire.

Il s'agit donc de l'armoire, tiroir, boîte, farde et poubelle. Ils forment l'environnement qui permet le rangement des informations.

Leur description et leur utilisation étant similaires à ce qui a été décrit au premier chapitre, ces types d'objet ne seront pas redéfinis.

2.3. LES TYPES D'OBJET LOGIQUES

Rappelons que les types d'objet logiques sont l'information et le groupe d'informations. Ils représentent les objets classés dans l'E.R.

Il est nécessaire de préciser ces types d'objet dans le cadre d'un E.R. automatisé (LESU85).

2.3.1. LA STRUCTURE GENERALE DE L'INFORMATION

2.3.1.1. L'INFORMATION

Définition

Une information est un ensemble structuré de signes ou symboles porteurs de connaissance pour celui qui les reçoit.

Rappelons que les modes de représentation de l'information sont divers, il a notamment les modes : oral, écrit, graphique ou mixte.

Les informations peuvent être des lettres, des formulaires, des résultats comptables, ... On n'est cependant pas obligé de limiter cette liste à des exemples d'informations faisant partie d'un bureau classique. On peut en effet l'étendre au milieu informatique. Seront donc considérés comme informations : la version source et la version code d'un programme, les données d'un programme, les résultats d'un programme, qu'ils soient de type texte, graphique, image ou de type message vocal digitalisé.

2.3.1.2. LA STRUCTURE DE L'INFORMATION

Quel que soit le mode de représentation, toute information est composée de trois grandes parties : l'en-tête, le corps et la clôture (figure 2-2) (LESU85) (ROBE85).

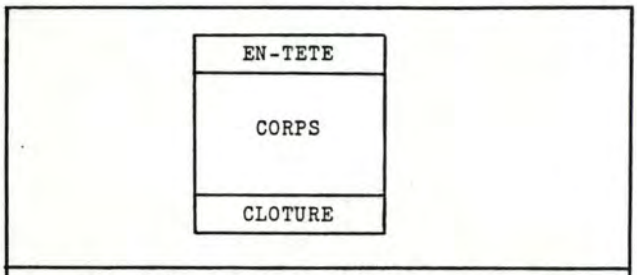


Fig. 2-2 : Structure d'une information

1. EN-TETE

L'en-tête d'une information comprend :

- son identifiant (normalement unique) :
  exemples : "CD/RL/015", "Note de service 84/16", ... ;
- son type : exemples : note de frais, bon de commande, ... ;
- le sujet de l'information par texte continu, mots-clés,..
  exemples :concerne : "votre note de service 84/16",
  mots-clés : message systems, office automation ;
- le nom du rédacteur qui permettra par exemple de retrouver tous les documents générés par un employé, un cadre, une secrétaire, ...
- un statut de confidentialité qui permet au rédacteur de se protéger contre toute indiscrétion.

2. CLOTURE

La clôture d'une information comprend :

- des renseignements concernant l'expédition éventuelle de l'information :
  - . le nom de l'expéditeur ;
  - . le nom du ou des destinataires ;
  - . la date d'expédition ;
  - . un témoin d'attente de réponse ;

. un témoin d'attente de confirmation de réception ;

- des renseignements concernant la réception éventuelle de l'information :

  . la date de réception ;

  . la liste des réponses ;

  . la liste des accusés de réception ;

- l'état de l'information est un concept relatif au cycle de vie d'une information. Une information peut être susceptible d'être révisée, signée, expédiée, classée ou archivée, s'il s'agit d'une lettre par exemple ;

  S'il s'agit d'un bon de commande, il peut être : reçu, vérifié, enregistré, en attente de livraison, en attente de facturation.

- la date du dernier changement d'état permet de connaître la date de classement ou d'archivage dans le cas où, par exemple, une politique spéciale est suivie pour l'archivage qui se baserait sur la date de classement.

3. LE CORPS

Il s'agit du contenu proprement dit de l'information.

Le corps de l'information comprend :

- la référence du support informatique où est stocké le contenu de l'information dans le cas où cette information serait digitalisée : exemple : la référence complète du fichier sur disque, bande, ...

- la référence de la localisation où le contenu de l'information originale est accessible.

  Cette deuxième référence peut exister en parallèle avec la première pour certaines informations où pour des raisons d'ordre juridique, l'original doit être conservé comme preuve en cas de conflit.

  Elle peut aussi représenter la seule référence à l'information dans le cas où celle-ci n'est pas sur support informatique. Par exemple, la localisation d'un ouvrage dans une bibliothèque ;

- la nature de l'information : on distinguera les documents, les messages et les formulaires.

A. LE DOCUMENT

Convenons d'appeler document, une information spécifique, généralement représentée sous forme écrite, graphique ou mixte et dont le corps est structuré logiquement et physiquement de la manière suivante :

- *structure logique*

La base de la structure logique d'un document est le paragraphe. A partir de cette entité, tout document peut être logiquement structuré sous forme d'arborescence (figure 2-3).

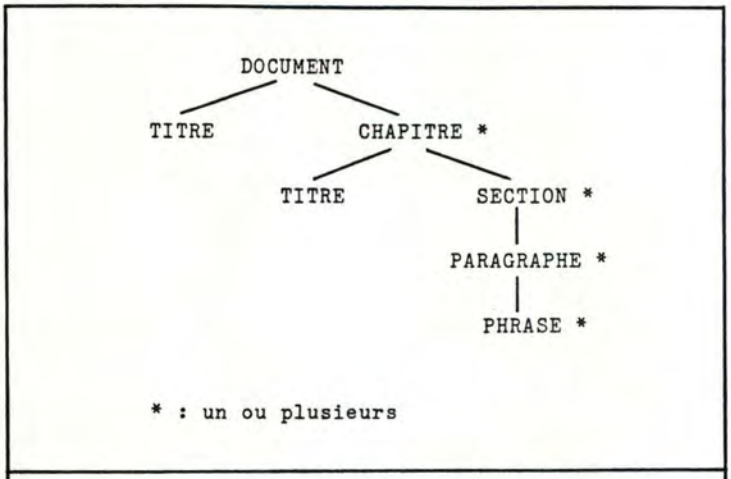


Fig. 2-3 : La structure logique d'un document

- *structure physique :*

Convenons d'appeler structure physique d'un document la forme de sa représentation externe au destinataire. On peut aussi la représenter au moyen d'une arborescence (figure 2-4).

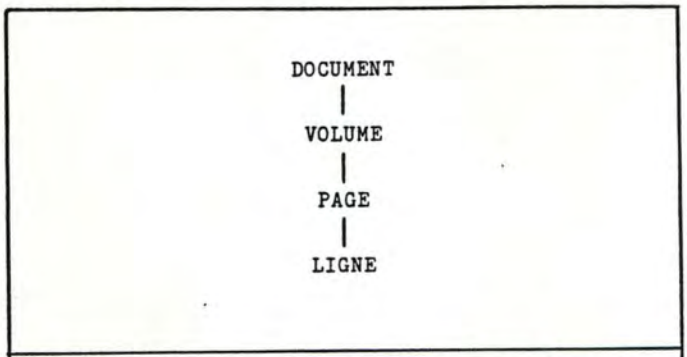


Fig. 2-4 : Structure physique d'un document

Une analyse détaillée d'une structure logique et physique de document est décrite dans (DAPA85).

B. LE MESSAGE

Convenons d'appeler message une information sous forme écrite/orale (généralement) qui a les caractéristiques suivantes :

- elle est, en général, brève ;

- le corps n'obéit à aucune structure particulière ;

- elle n'est généralement ni classée ni archivée.

C. LE FORMULAIRE

Convenons d'appeler formulaire une information écrite standardisée dont le corps est structuré logiquement et physiquement de la manière suivante :

- structure logique :

Un formulaire est constitué d'un squelette textuel et d'un nombre fixe de champs auxquels peuvent être associés des ensembles de valeurs (figure 2-5).
Exemples : Bons de commande, formulaire de demande de renseignements ...

BON DE COMMANDE :

- numéro du bon
- date de commande
- pour chaque article commandé :
  - . intitulé de l'article
  - . quantité commandée
  - . prix unitaire
  - . prix total
- prix total de la commande

Fig. 2-5 : Structure logique d'un formulaire

- structure physique :

La structure physique d'un formulaire, tout comme celle d'un document, est la forme de la présentation externe du formulaire pour les destinataires et expéditeurs (figure 2-6).

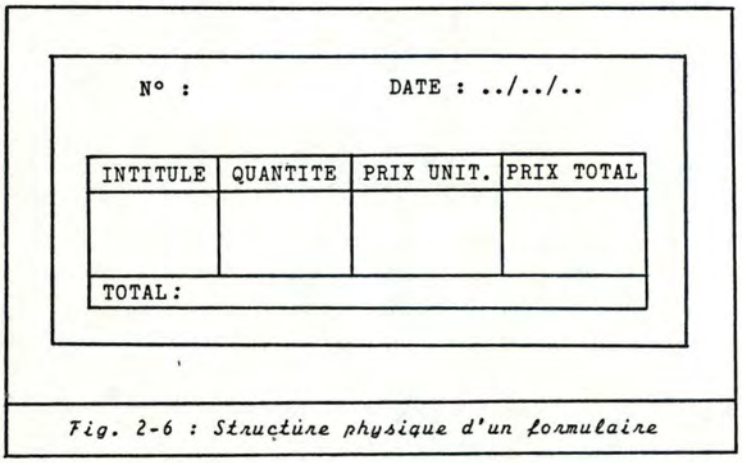


Fig. 2-6 : Structure physique d'un formulaire

A une même structure logique peuvent correspondre diverses structures physiques de formulaire.

Une analyse détaillée et une proposition d'implémentation sur un Système de Gestion de Base de Données Relationnel (SGBDR) du concept de formulaire sont présentés dans (BHSL84).

2.3.2. ORGANISATION DE L'INFORMATION

Les informations de l'E.R. sont strucurées en groupes d'informations en vue d'une facilité de la gestion de leur utilisation ultérieure. Il s'agit du concept de groupe d'informations.

On distingue trois groupements possibles :

1. Les fichiers

Les fichiers sont des groupes d'occurrences d'un même type de formulaire (quelque fois de message ou de document) univoquement identifiés et arrangés selon un certain ordre (alphabétique, chronologique).

Exemples : Fichier des bons de commande,
Fichier des notes de service 1984,
Fichier des TLX mois mars,
Fichier courrier mois en cours, ...

2. Les dossiers

Les dossiers sont des groupes de plusieurs types différents de formulaire /message /document reliés entre eux par une relation logique.

Exemple : Dossier médical,
Dossier assurance vie, ...

3. Les piles

Les piles sont des groupes d'occurrences de divers types de document/message/formulaire sans lien logique entre eux, sauf généralement l'ordre chronologique inverse de leur ordre d'arrivée. Mais cet ordre n'est pas intentionnel.

Ce sont des informations en attente de classement, de traitement (courrier IN, OUT).

3. LES OPERATIONS SUR LES OBJETS DE L'E.R.

Pour être valide, toute opération sur un objet doit être demandée par un utilisateur reconnu comme tel par l'E.R. D'autres conditions sont exigées suivant le type d'opération.

1. La création

Il doit être possible d'ajouter de nouvelles informations dans l'E.R., de même que de créer de nouveaux objets pour permettre le rangement des informations.

Certaines conditions doivent cependant être remplies non seulement par l'utilisateur qui demande la création mais aussi par l'objet qui est demandé à être créé. Par exemple, l'utilisateur demandeur doit être habilité à créer de nouveaux objets.

2. La suppression

Il doit également être possible de détruire certaines informations en les mettant dans la poubelle ou d'enlever certains objets de rangement.

A nouveau, certaines conditions doivent être remplies. Par exemple, l'utilisateur qui demande la suppression d'une information doit en être le propriétaire.

3. La modification

Il doit être possible de déplacer une information d'un endroit vers un autre, de changer certaines propriétés telles que l'état, le type ou le sujet d'une information.

4. La consultation

Un utilisateur doit pouvoir consulter le contenu d'une information avec la sécurité que durant tout le temps de la consultation aucune modification ne survienne à celle-ci. Rien n'empêche néanmoins qu'une même information soit consultée simultanément par différents utilisateurs.

5. La mise à jour

Un utilisateur doit pouvoir mettre à jour le contenu d'une information. Mais cette information ne doit pas être durant ce temps en consultation ou en mise à jour par un autre utilisateur. Pendant une mise à jour, l'information est complètement inaccessible à tout autre utilisateur.

6. La recherche

Un utilisateur doit pouvoir rechercher les informations correspondant à un ou plusieurs critères de son choix. Ces critères portent sur les propriétés des informations. Par exemple, rechercher toutes les informations d'un certain type et répondant à certains mots-clés.

4. CONCLUSION

Ce second chapitre a servi de première approche de l'E.R. d'informations automatisé par comparaison à l'E.R. manuel décrit au premier chapitre.

Il a notamment permis d'approfondir les concepts suivants :

- l'utilisateur et ses exigences ;
- le groupe d'utilisateurs, son origine et son organisation en hiérarchie ;
- une méthode d'accès aux informations basée sur la gestion de la liste des groupes d'utilisateurs ayant accès aux informations ;
- la structure de l'information en en-tête, corps et clôture ;
- les types d'information : document, message et formulaire ;
- les regroupements possibles d'informations sous forme de fichiers, dossiers et piles ;
- et enfin, un survol des opérations sur l'E.R.

CHAPITRE 3 : ANALYSE FONCTIONNELLE DE L'ENVIRONNEMENT DE RANGEMENT D'INFORMATIONS

1. INTRODUCTION

Dans le deuxième chapitre, nous avons identifié les objets et les opérations que 1'on désirerait retrouver dans un modèle de rangement d'informations.

Ce troisième chapitre est consacré à une modélisation plus formelle de 1'E.R. qui sera implémenté.

Il se présentera sous deux parties principales où 1'on verra d'une part les objets et d'autre part les opérations.

Dans la première partie sera dressée une présentation des types d'objet en utilisant le modèle Entité-Association (E-A) (BOPI83). Les types d'entité (T.E.) seront spécifiés par leurs attributs, les associations ainsi que les contraintes d'intégrité du schéma.

La deuxième partie spécifiera les opérations par leurs données, résultats, traitement ainsi que leurs préconditions et leurs postconditions.

2. LES OBJETS DE L'ENVIRONNEMENT DE RANGEMENT (E.R.)

2.1. Le schéma Entité-Association (E-A) de 1'E.R.

Le modèle E-A a été choisi comme modèle conceptuel de structuration des types d'objet retenus dans le deuxième chapitre. Chaque type d'objet y est représenté par un T.E. Les relations de composition existant entre les types d'objet y sont représentés par des associations. Certaines propriétés des types d'objet ne peuvent pas être représentées par ce modèle. Il doit pour cette raison être complété par un certain nombre de contraintes d'intégrité (C.I.).

La figure 3-1 illustre le schéma E-A de 1'E.R.

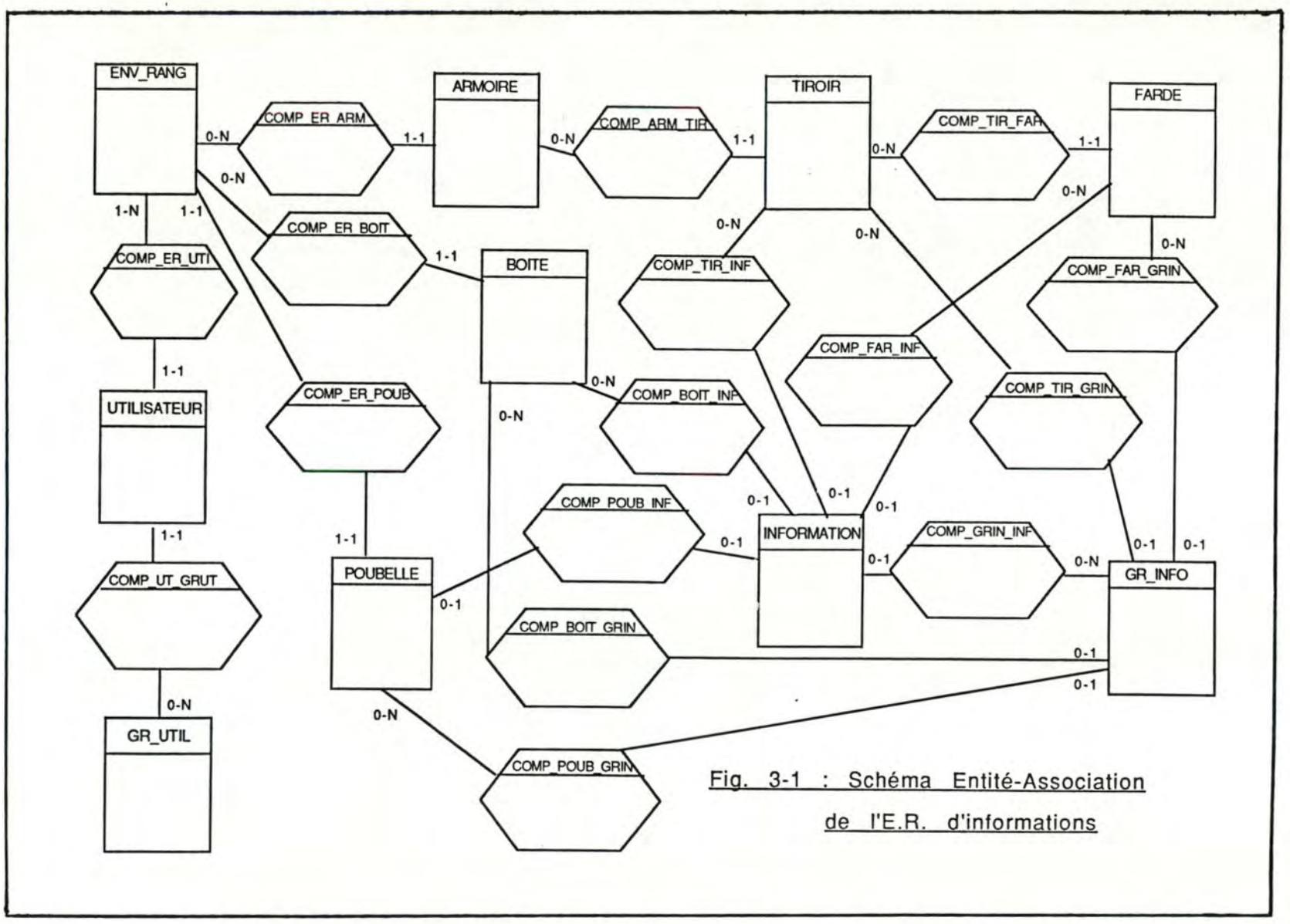


Fig. 3-1 : Schéma Entité-Association de l'E.R. d'informations

2.2. Interprétation du schéma E-A

L'interprétation du schéma E-A de l'E.R. consistera en l'analyse des types d'entité (T.E.) ainsi que de leurs attributs, des associations et des contraintes d'intégrités.

2.2.1. Les types d'entité

Dans le schéma E-A (figure 3-1), les T.E. sont représentés graphiquement par des rectangles (figure 3-2). Chaque T.E. possède un nom ainsi qu'un certain nombre d'attributs.

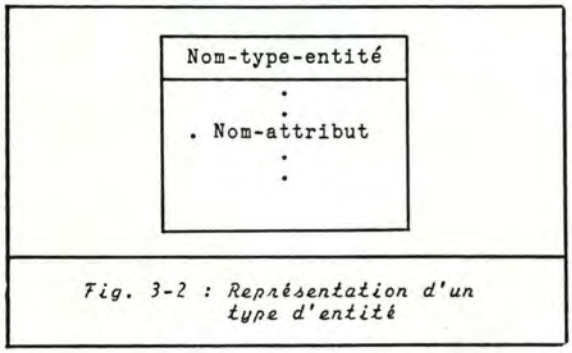


Fig. 3-2 : Représentation d'un type d'entité

A. Liste des T.E. :

- *UTILISATEUR*
- *GROUPE-UTILISATEUR*
- *ENVIRONNEMENT-RANGEMENT*
- *ARMOIRE*
- *TIROIR*
- *FARDE*
- *POUBELLE*
- *BOITE*
- *INFORMATION*
- *GROUPE-INFORMATION*

B. Interprétation

La signification de chacun des T.E. ne sera pas donnée, ceux-ci ayant fait l'objet des deux chapitres précédents. Nous avons simplement ajouté le T.E. ENVIRONNEMENT-RANGEMENT pour généraliser la relation de composition et permettre ainsi de relier tous les T.E. entre eux. Dans la version actuelle du modèle, il n'existe cependant jamais qu'une et une seule occurrence du T.E. ENVIRONNEMENT-RANGEMENT. Dans une version ultérieure on pourrait considérer que le modèle puisse gérer plusieurs E.R. De même, il n'existera qu'une occurrence du T.E. POUBELLE. Nous considérons en effet qu'une poubelle suffit pour un E.R.

2.2.2. Les attributs des T.E.

Dans le schéma E-A (figure 3-1), les noms des attributs sont notés à l'intérieur des T.E. (figure 3-2). Les attributs soulignés sont des identifiants globaux et locaux. Les attributs entre parenthèses sont des attributs répétitifs.

A. Liste des attributs par T.E. :

UTILISATEUR :

- *NOM*
- *CODE*
- *MOT-DE-PASSE*
- *DATE-DE-CREATION*
- *CREATEUR*
- *(PERMISSION-D'ACCES)*

GROUPE-UTILISATEUR :

- *NOM*
- *CODE*
- *DATE-DE-CREATION*
- *CREATEUR*

ENVIRONNEMENT-RANGEMENT :

- *NOM*
- *CODE*
- *DATE-DE-CREATION*
- *CREATEUR*
- *DESCRIPTION*
- *STATUT-DE-CONFIDENTIALITE*
- *(POSSIBILITE-D'ACCES)*

POUBELLE :

- *NOM*
- *CODE*
- *DATE-DE-CREATION*
- *CREATEUR*
- *DESCRIPTION*
- *STATUT-DE-CONFIDENTIALITE*
- *(POSSIBILITE-D'ACCES)*

BOITE :

- *NOM*
- *CODE*
- *DATE-DE-CREATION*
- *CREATEUR*
- *DESCRIPTION*
- *STATUT-DE-CONFIDENTIALITE*

. TYPE
. (POSSIBILITE-D'ACCES)

ARMOIRE :

. NOM
. CODE
. DATE-DE-CREATION
. CREATEUR
. DESCRIPTION
. STATUT-DE-CONFIDENTIALITE
. TYPE
. (POSSIBILITE-D'ACCES)

TIROIR :

. NOM
. CODE
. DATE-DE-CREATION
. CREATEUR
. DESCRIPTION
. STATUT-DE-CONFIDENTIALITE
. TYPE
. (POSSIBILITE-D'ACCES)

FARDE :

. NOM
. CODE
. DATE-DE-CREATION
. CREATEUR
. DESCRIPTION
. STATUT-DE-CONFIDENTIALITE
. TYPE
. (POSSIBILITE-D'ACCES)

GROUPE-INFORMATION :

. NOM
. CODE
. DATE-DE-CREATION
. CREATEUR
. PRESENCE
. DESCRIPTION
. STATUT-DE-CONFIDENTIALITE
. TYPE
. (POSSIBILITE-D'ACCES-CONSULTATION)
. (POSSIBILITE-D'ACCES-MISE-A-JOUR)

INFORMATION :

. NOM
. CODE
. NATURE
. DATE-DE-CREATION

. CREATEUR
. PRESENCE
. DATE-DE-CONSULTATION
. CONSULT
. DATE-DE-MISE-A-JOUR
. MODIFICAT
. DESCRIPTION
. STATUT-DE-CONFIDENTIALITE
. TYPE
. (POSSIBILITE-D'ACCES-CONSULTATION)
. (POSSIBILITE-D'ACCES-MISE-A-JOUR)
. EXPEDITION
. MOTS-CLES)
. REFERENCE-STOCKAGE
. REFERENCE-ORIGINAL

B. Interprétation

La liste de tous les attributs pour chaque type d'article (T.A.) ne sera pas analysée selon l'ordre dans lesquels ils ont été présentés ci-dessus. On peut en effet remarquer une grande similarité entre certains attributs de T.E. différents. Par exemple, chaque T.E. possède un code, un nom, un créateur, une date de création, ... En regardant de plus près cette similarité, on peut voir apparaître 3 groupes de T.E. :

- un premier groupe se compose des T.E. : ENVIRONNEMENT-RANGEMENT, ARMOIRE, TIROIR, FARDE, POUBELLE et BOITE, qui correspondent aux types d'objet physiques selon la définition donnée dans le deuxième chapitre ;

- un deuxième groupe se compose des T.E. : INFORMATION et GROUPE-INFORMATION qui correspondent aux types d'objet logiques ;

- un troisième groupe se compose des T.E. : UTILISATEUR et GROUPE-UTILISATEUR.

Les attributs seront donc analysés par groupe de T.E. Pour chacun des groupes, nous reprendrons l'ensemble des attributs de chaque T.E. faisant partie du groupe en rejetant ceux qui reviennent plusieurs fois. Nous obtiendrons ainsi l'union des attributs des T.E. du groupe. Lors de l'analyse de chacun des attributs nous donnerons une définition générale pour le groupe et nous mentionnerons si nécessaire les particularités pour certains T.E., par exemples si cet attribut n'existe pas pour un ou plusieurs T.E. du groupe ou s'il diffère par rapport à la définition générale.

Avant de passer à cette analyse, décrivont les propriétés qui formeront la définition générale des attributs.

C. Définition des propriétés des attributs

Les concepts utilisés dans l'analyse des attributs des T.E. sont principalement repris de (BOPI83). Des concepts supplémentaires ont cependant été ajoutés en vue d'une plus grande précision dans l'analyse. Ces concepts sont inspirés des propriétés des items du modèle d'accès dans (HAIN81).

Chaque attribut d'un T.E. possède un certain nombre de propriétés qui permettent de déterminer le rôle de l'attribut dans le T.E. ainsi que son mode d'utilisation.

Liste des propriétés d'un attribut :

- IDENTIFIANT ou NON IDENTIFIANT
- SIMPLE ou REPETITIF
- ELEMENTAIRE ou DECOMPOSABLE
- OBLIGATOIRE ou FACULTATIF
- DEFINITIF ou MODIFIABLE
- AUTOMATIQUE ou MANUEL
- TYPE DE VALEUR

1. Attribut IDENTIFIANT ou NON IDENTIFIANT

Un identifiant peut être vu comme une contrainte d'intégrité comme décrite dans (BOPI83). Nous avons cependant préféré le noter comme propriété d'un attribut. Ce choix est particulier à notre schéma E-A où les identifiants sont constitués d'un seul attribut.

Rappelons qu'un identifiant d'un T.E. permet de repérer univoquement chaque occurrence de ce type. De plus, un T.E. peut être doté de plus d'un identifiant.

Nous distinguerons des identifiants globaux et des identifiants locaux. Un identifiant d'un T.E. est global s'il permet de repérer univoquement chaque occurrence de ce type dans l'ensemble du schéma E-A. Il sera local s'il ne permet de repérer univoquement chaque occurrence de ce type que dans un espace limité du schéma E-A. Dans ce cas cet espace devra être spécifié.

2. Attribut SIMPLE ou REPETITIF

"Un attribut est simple si pour une occurrence d'un T.E., il ne peut prendre qu'une seule valeur."

"Il est répétitif si pour une occurrence d'un T.E., il peut prendre plusieurs valeurs d'un même type."

exemple : PRENOMS-PERSONNE.

3. Attribut ELEMENTAIRE ou DECOMPOSABLE

"Un attribut est décomposable si à une occurrence d'un T.E., il fait correspondre un groupe de valeurs de types différents et peut être décomposé, au plus, en autant d'attributs qu'il y a de types différents dans le groupe de valeurs."

"Un attribut non décomposable est dit élémentaire."

L'attribut ADRESSE-DE-PERSONNE prenant valeur dans le groupe (NR-D'HABITATION, NOM-RUE, CODE-POSTAL, NOM-COMMUNE) peut-être décomposé en 4 attributs :

- *NR-D'HABITATION-DE-PERSONNE*
- *NOM-RUE-DE-PERSONNE*
- *CODE-POSTAL-DE-PERSONNE*
- *NOM-COMMUNE-DE-PERSONNE*

Notons qu'un attribut peut-être à la fois décomposable et répétitif.

Exemple : l'attribut RESULTATS-EXAMENS pourrait être défini comme un groupe répétitif du T.E. ETUDIANT.

4. Attribut OBLIGATOIRE ou FACULTATIF - VALEUR INEXISTANTE

"Introduisons cette propriété par l'analyse d'un exemple. Soit un système d'information où l'on a défini le T.E. PERSONNE caractérisé par les attributs NOM-DE-FAMILLE, NOM-DE-JEUNE-FILLE, PRENOM, ADRESSE-PERSONNE et SEXE."

"Dans la définition de ce type, il est normal de considérer que toute personne possède un nom de famille, un ou plusieurs prénoms, une adresse et un sexe. Ces attributs de PERSONNE sont obligatoires : ils doivent prendre une valeur effective pour chaque occurrence du type qu'ils caractérisent. Par contre, l'attribut NOM-DE-JEUNE-FILLE ne prendra une valeur que pour les personnes mariées de sexe féminin. Cet attribut est facultatif car il peut ne pas prendre de valeur pour certaines occurrences du type qu'il caractérise ; cet attribut n'a pas de signification pour ces occurrences."

Valeur inconnue

"Lorsqu'une valeur d'un attribut existe pour une occurrence d'un type donné mais est inconnue à une date d'observation, on attribuera à cette occurrence la valeur inconnue. Afin de tenir compte de cette éventualité, on introduira la valeur "INCONNUE" comme élément de toute classe de valeur."

5. Attribut DEFINITIF ou MODIFIABLE

Cette propriété a été ajoutée par rapport à (BOPI83) pour donner un sens plus restrictif dans l'utilisation de certains attributs.

Un attribut sera définitif si pour une occurrence d'un T.E., cet attribut ne peut recevoir de valeur qu'une seule fois et ceci lors de la création de l'occurrence du T.E.

Il sera modifiable si pour une occurrence d'un T.E., cet attribut peut changer de valeur après la création de l'occurrence du T.E.

6. Attribut AUTOMATIQUE ou MANUEL

Un attribut est automatique si pour une occurrence d'un T.E., la valeur de cet attribut peut-être déterminée automatiquement.

exemple : CODE-OBJET est généré automatiquement en incrémentant un compteur.

Il est manuel si pour une occurrence d'un T.E., sa valeur ne peut être déterminée que par l'intervention de l'utilisateur.

exemple : NOM-OBJET représente le nom que l'utilisateur a décidé de donner à l'objet pour l'identifier.

7. TYPE DE VALEUR

"Le type de valeur représente la classe de toutes les valeurs possibles qui vérifient la définition du type ; cette définition peut être donnée soit par la liste des éléments de la classe, soit par la propriété que les membres de la classe doivent satisfaire."

exemples :

- *NOM-OBJET = chaîne finie de caractères alphanumériques ;*
- *CODE-OBJET = nombre entier ;*
- *PRESENCE-OBJET = (PRESENT, EN-CONSULTATION, EN-MISE-A-JOUR).*

D. Attributs des T.E. du groupe : ENVIRONNEMENT-RANGEMENT, ARMOIRE, TIROIR, FARDE, POUBELLE et BOITE

Pour que les interprétations qui seront données puissent correspondre à chacun des T.E. de ce groupe, nous parlerons plus généralement d'"objet".

Liste des attributs des T.E. du groupe :

- *NOM*
- *CODE*
- *DATE-DE-CREATION*
- *CREATEUR*

. DESCRIPTION
. STATUT-DE-CONFIDENTIALITE
. TYPE
. POSSIBILITE-D'ACCES

1. NOM :

. Signification : l'attribut NOM d'un objet permettra d'identifier cet objet parmi les autres objets de ce type faisant partie de l'E.R.

. Propriétés :

- IDENTIFIANT LOCAL
- SIMPLE
- ELEMENTAIRE
- OBLIGATOIRE
- MODIFIABLE
- MANUEL
- TYPE DE VALEUR : chaîne finie de caractères alphanumériques.

  exemples :
  "POUBELLE" pour POUBELLE,
  "IN", "OUT", "URGENT" pour BOITE

a. Identifiant local

La portée de l'identifiant s'étend à tout le schéma pour les T.E. : ENVIRONNEMENT-RANGEMENT, ARMOIRE, POUBELLE et BOITE.

Pour le T.E. TIROIR, la portée est limitée à l'armoire à laquelle le tiroir appartient. Cela signifie que deux tiroirs appartenant à des armoires différentes peuvent porter le même nom.

Pour le T.E. FARDE, la portée est limitée au tiroir auquel la farde appartient.

b. Obligatoire

Tout objet de ce groupe reçoit un NOM lors de sa création.

Ceci est également vrai pour les objets des deux autres groupes.

c. Modifiable

Le problème de la modifiabilité d'un attribut sera de distinguer les différents types d'utilisateur pouvant opérer la modification.

Nous distinguerons pour cela quatre classes d'utilisateurs :

- *le propriétaire de l'objet ;*
- *l'administrateur de l'E.R. ;*
- *les utilisateurs ayant accès à l'objet ;*
- *les utilisateurs ne correspondant à aucune des trois premières classes.*

Remarques :

1) Il ne peut y avoir pour chaque objet qu'un seul propriétaire. Il s'agit en fait de l'utilisateur qui a demandé sa création.

2) Il n'y a qu'un administrateur de l'E.R.

	Propriétaire	Administrateur	Utilisateur accès autorisé	Autres
OBJET	X	X	-	-

X : peut modifier l'attribut
- : ne peut pas modifier l'attribut

Fig. 3-3 : Possibilité de modification de la valeur d'un attribut

La figure 3-3 illustre la possibilité de modification qui servira de référence à tous les attributs modifiables de ce présent groupe d'objets.

d. Manuel

L'attribution d'un nom à un objet étant manuelle et seul l'administrateur de l'E.R. étant actuellement (1) autorisé à créer des objets de ce groupe, c'est lui qui décidera du nom de l'objet. Dans ce cas l'administrateur joue à la fois le rôle de propriétaire.

Cette restriction n'est cependant pas d'application pour le T.E. FARDE. Une farde peut être créée par un utilisateur autre que l'administrateur. Dans ce cas, c'est le propriétaire de la farde qui décidera du nom de celle-ci.

2. CODE :

. Signification : le CODE est un identificateur interne qui permettra au système de gérer les objets. Il servira notamment d'accéder aux objets.

. Propriétés :

- IDENTIFIANT GLOBAL
- SIMPLE
- ELEMENTAIRE
- OBLIGATOIRE
- DEFINITIF
- AUTOMATIQUE
- TYPE DE VALEUR : nombre entier.
  exemples : 00023, 12478, 00007.

a. Définitif

Le CODE d'un objet ne peut plus être modifié après la création de celui-ci.

b. Automatique

Le CODE d'un objet est généré automatiquement à partir d'un compteur qui est incrémenté de 1 après chaque création d'un objet.

3. DATE-DE-CREATION :

. Signification : la DATE-DE-CREATION représente la date à laquelle l'objet a été créé dans l'E.R.

---

(1) Le fait que seul l'administrateur de l'E.R. puisse créer, modifier et supprimer des objets physiques est une décision prise pour une première version de l'application. Tous tous les attributs nécessaires on cependant été prévu pour étendre le droit à d'autres utilisateurs ultérieurement. Par exemples : CREATEUR, DATE-DE-CREATION, CONSULT, DATE-DE-CONSULTATION, ...

. *Propriétés* :

- *NON IDENTIFIANT*
- *SIMPLE*
- *DECOMPOSABLE*
- *OBLIGATOIRE*
- *DEFINITIF*
- *AUTOMATIQUE*
- *TYPE DE VALEUR : DATE*
  exemples : 30-11-85 ; 21-01-86.

a. Décomposable

La DATE-DE-CREATION d'un objet est décomposable en : jour, mois et année.

b. Automatique

La DATE-DE-CREATION d'un objet est calculée à partir de la date du système.

4. CREATEUR :

. *Signification : le CREATEUR de l'objet correspond à l'utilisateur qui a demandé la création de cet objet. Il représente le CODE de cet utilisateur.*

. *Propriétés* :

- *NON IDENTIFIANT*
- *SIMPLE*
- *ELEMENTAIRE*
- *OBLIGATOIRE*
- *DEFINITIF*
- *AUTOMATIQUE*
- *TYPE DE VALEUR : nombre entier.*
  exemples : 00023, 12478, 00007.

a. Automatique

Le CREATEUR d'un objet est généré automatiquement à partir du CODE de l'utilisateur qui a demandé la création de l'objet.

5. DESCRIPTION :

. *Signification : l'attribut DESCRIPTION d'un objet est un commentaire que le propriétaire peut lier à l'objet.*

. *Propriétés* :

- *NON IDENTIFIANT*
- *SIMPLE*
- *ELEMENTAIRE*

- FACULTATIF
- MODIFIABLE
- MANUEL
- TYPE DE VALEUR : chaîne finie de caractères alphanumériques.
  exemple : "cet objet contient des renseignements sur le sujet X".

6. STATUT-DE-CONFIDENTIALITE :

. Signification : le STATUT-DE-CONFIDENTIALITE est une possibilité donnée au propriétaire de l'objet d'empêcher quiconque d'autre que lui d'avoir accès à l'objet.

. Propriétés :

- NON IDENTIFIANT
- SIMPLE
- ELEMENTAIRE
- OBLIGATOIRE
- MODIFIABLE
- AUTOMATIQUE
- TYPE DE VALEUR : (CONFIDENTIEL, NON CONFIDENTIEL)

a. Automatique

Le STATUT-DE-CONFIDENTIALITE prend la valeur "NON CONFIDENTIEL" par défaut.

7. TYPE :

. Signification : l'attribut TYPE permet la classification des objets selon un critère choisi par le propriétaire. Il peut éventuellement convenir d'une nomenclature déterminée avec d'autres utilisateurs. Le type a pour but de faciliter la recherche des objets. On verra que la fonction RECHERCHE permettra de rechercher des objets sur base de leur type.

Cet attribut n'existe pas pour les T.E. ENVIRONNEMENT-RANGEMENT et POUBELLE. Il n'aurait, en effet, pas de raison d'être puisque nous limitons notre modèle à un E.R. et à une poubelle.

. Propriétés :

- NON IDENTIFIANT
- SIMPLE
- ELEMENTAIRE
- FACULTATIF
- MODIFIABLE
- MANUEL

- TYPE DE VALEUR : chaîne finie de
  caractères alphanumériques.
  exemples : "PROJET ALPHA",
  "INSTITUT D'INFORMATIQUE".

8. POSSIBILITE-D'ACCES :

. Signification : l'attribut POSSIBILITE-D'ACCES d'un objet permet au propriétaire de l'objet de fixer la liste des groupes d'utilisateurs qui peuvent avoir accès à celui-ci.

L'accès par les utilisateurs faisant partie de cette liste, autres que le propriétaire et l'administrateur de l'E.R., ne sera cependant permis que si l'attribut STATUT-DE-CONFIDENTIALITE a la valeur "NON CONFIDENTIEL". En effet, dans le cas ou cet attribut a pour valeur "CONFIDENTIEL", seul le propriétaire a encore accès à l'objet.

Cette liste de groupes d'utilisateurs reprendra les noms de ces groupes.

Pour les T.E. ENVIRONNEMENT-RANGEMENT et POUBELLE, l'attribut POSSIBILITE-D'ACCES prendra comme valeur particulière la liste de tous les groupes d'utilisateurs. Nous considérons, en effet, que ces objets sont accessibles à tous.

. Propriétés :

- NON IDENTIFIANT
- REPETITIF
- ELEMENTAIRE
- OBLIGATOIRE
- MODIFIABLE
- MANUEL
- TYPE DE VALEUR : chaîne finie de
  caractères alphanumériques.
  exemples : "Groupe-bureautique",
  "Groupe-O.S.", ...

a. Répétitif

Liste des groupes d'utilisateurs qui peuvent accéder à l'objet.

E. Attributs des T.E. du groupe : INFORMATION et GROUPE-INFORMATION :

Dans cette partie, nous utiliserons à nouveau le terme "objet" pour INFORMATION ou GROUPE-INFORMATION sauf s'il est spécifié autrement.

Liste des attributs du groupe :

- NOM
- CODE
- NATURE
- DATE-DE-CREATION
- CREATEUR
- PRESENCE
- DATE-DE-CONSULTATION
- CONSULT
- DATE-DE-MISE-A-JOUR
- MODIFICAT
- DESCRIPTION
- STATUT-DE-CONFIDENTIALITE
- TYPE
- POSSIBILITE-D'ACCES-CONSULTATION
- POSSIBILITE-D'ACCES-MISE-A-JOUR
- EXPEDITION
- MOTS-CLES
- REFERENCE-STOCKAGE
- REFERENCE-ORIGINAL

Remarque :

Les remarques qui ont été établies concernant les attributs des types d'objet du groupe précédant, c'est-à-dire les types d'objet : environnement de rangement, armoire, tiroir, farde poubelle et boîte, sont applicables aux types d'objet information et groupe-information sauf s'il est spécifié autrement.

1. NOM :

. Signification : l'attribut NOM d'un objet permettra d'identifier cet objet parmi les autres objets de ce type faisant partie de l'E.R.

. Propriétés :

- IDENTIFIANT LOCAL
- SIMPLE
- ELEMENTAIRE
- OBLIGATOIRE
- MODIFIABLE
- MANUEL

- TYPE DE VALEUR : chaîne finie de
caractères alphanumériques.
exemple : "Formulaire-Inscription-85-86".

a. Identifiant local

La portée de l'identifiant d'une information ou d'un groupe d'informations est limitée à l'objet dans lequel cet identifiant est contenu. Cet objet peut être du type : GROUPE-INFORMATION, FARDE, TIROIR, BOITE, POUBELLE.

Le NOM de l'information ou du groupe d'informations est donc unique dans l'objet dans lequel il est contenu. Par exemple, deux informations d'une même farde devront porter des noms différents.

b. Modifiable

Rappelons que dans le premier groupe d'objets, on avait distingué quatre classes d'utilisateurs : le propriétaire de l'objet, l'administrateur de l'E.R., les utilisateurs ayant accès à l'objet et les autres.

Dans le groupe d'objets de la présente section, la propriété d'accès sera décomposée en deux classes. On distinguera les utilisateurs qui peuvent accéder l'objet pour le mettre à jour et d'autres qui ne pourront que le consulter.

Il y aura donc cinq classes d'utilisateurs :

. le propriétaire de l'objet ;

. l'administrateur de l'E.R. ;

. les utilisateurs ayant accès à l'objet en consultation uniquement ;

. les utilisateurs ayant accès à l'objet en mise à jour donc automatiquement en consultation ;

. les utilisateurs ne correspondant à aucune des quatre premières classes.

La figure 3-4 illustre la possibilité de modification qui servira de référence à tous les attributs modifiables de ce présent groupe d'objets.

	Propriétaire	Administrateur	Utilisateur accès autorisé en consultation	Utilisateur accès autorisé en mise à jour	Autres
OBJET	X	X	-	-	-

X : peut modifier l'attribut
- : ne peut pas modifier l'attribut

Fig. 3-4 : Possibilité de modification de la valeur d'un attribut

c. Manuel

L'attribution du NOM de l'objet se fait lors de la création de l'objet.

Tout utilisateur peut créer un objet, s'il en a la permission, en l'associant à une structure de rangement à laquelle il a accès. Par exemple, il peut créer une information dans une farde à laquelle il a accès.

2. CODE :

La signification et les propriétés de l'attribut CODE sont les mêmes que celles énoncées pour le premier groupe d'objets : ENV-RANG, ARMOIRE, TIROIR, POUBELLE et BOITE.

3. NATURE :

. Signification : *l'attribut NATURE n'est utilisé que pour le T.E. INFORMATION. Il permet de distinguer les informations de type : document, message et formulaire.*

. Propriétés :

- *NON IDENTIFIANT*
- *SIMPLE*
- *ELEMENTAIRE*

- OBLIGATOIRE
- MODIFIABLE
- MANUEL
- TYPE DE VALEUR : (DOCUMENT, MESSAGE, FORMULAIRE).

4. DATE-DE-CREATION :

La signification et les propriétés des attributs DATE-DE-CREATION et CREATEUR sont les mêmes que celles énoncées pour le premier groupe d'objets : ENV-RANG, ARMOIRE, TIROIR, ...

5. CREATEUR :

6. PRESENCE :

. Signification : l'attribut PRESENCE a pour mission d'indiquer si l'objet est actuellement présent dans l'E.R. où si quelqu'un l'a "sorti".

On peut sortir un objet de l'E.R. pour deux raisons :

1. Simplement pour le consulter. Il est préférable de ne pas permettre la modification d'un objet pendant sa consultation pour éviter une incohérence entre le renseignement obtenu par la consultation et l'état réel de l'objet.

2. Ou pour y opérer une mise à jour. Dans ce cas, on aimerait pouvoir prévenir les autres utilisateurs de ne pas consulter cette information temporairement pour la même raison que celle évoquée dans le point précédent.

En conclusion :

- un objet présent peut être "sorti" pour consultation ou mise à jour par tout utilisateur ayant ce droit ;

- un objet "sorti" en consultation ne sera plus disponible qu'en consultation ;

- un objet "sorti" en mise à jour ne sera plus disponible ni en consultation ni en mise à jour.

A la création de l'information, cet attribut a pour valeur "PRESENT".

. Propriétés :

- NON IDENTIFIANT
- SIMPLE
- ELEMENTAIRE
- OBLIGATOIRE

- MODIFIABLE
- AUTOMATIQUE
- TYPE DE VALEUR :
  (PRESENT,EN-CONSULTATION,EN-MISE-A-JOUR).

7. DATE-DE-CONSULTATION :

. Signification : la DATE-DE-CONSULTATION est la date à laquelle cet objet a été consulté pour la dernière fois.

A la création de l'objet, cet attribut prend la même valeur que la DATE-DE-CREATION de l'objet.

Cet attribut ne s'applique qu'au T.E. INFORMATION.

. Propriétés :

- NON IDENTIFIANT
- SIMPLE
- DECOMPOSABLE
- OBLIGATOIRE
- DEFINITIF
- AUTOMATIQUE
- TYPE DE VALEUR : DATE.

8. CONSULT :

. Signification : le CONSULT est l'utilisateur qui a consulté l'objet pour la dernière fois.

A la création de l'objet, cet attribut prend la même valeur que le CREATEUR de l'objet.

Cet attribut ne s'applique qu'au T.E. INFORMATION.

. Propriétés :

- NON IDENTIFIANT
- SIMPLE
- ELEMENTAIRE
- OBLIGATOIRE
- DEFINITIF
- AUTOMATIQUE
- TYPE DE VALEUR : nombre entier.
  exemples : 00023, 12478, 00007.

9. DATE-DE-MISE-A-JOUR :

. Signification : la DATE-DE-MISE-A-JOUR est la date à laquelle cet objet a été mis à jour pour la dernière fois.

A la création de l'objet, cet attribut prend la même valeur que la DATE-DE-CREATION de l'objet.

Cet attribut ne s'applique qu'au T.E. INFORMATION.

. *Propriétés* :

- *NON IDENTIFIANT*
- *SIMPLE*
- *DECOMPOSABLE*
- *OBLIGATOIRE*
- *DEFINITIF*
- *AUTOMATIQUE*
- *TYPE DE VALEUR : DATE.*

10. MODIFICAT :

. *Signification* : *le MODIFICAT est l'utilisateur qui a mis à jour l'objet pour la dernière fois.*

A la création de l'objet, cet attribut prend la même valeur que le CREATEUR de l'objet.

Cet attribut ne s'applique qu'au T.E. INFORMATION.

. *Propriétés* :

- *NON IDENTIFIANT*
- *SIMPLE*
- *ELEMENTAIRE*
- *OBLIGATOIRE*
- *DEFINITIF*
- *AUTOMATIQUE*
- *TYPE DE VALEUR : nombre entier.*
  exemples : 00023, 12478, 00007.

11. DESCRIPTION :

La signification et les propriétés des attributs DESCRIPTION, STATUT-DE-CONFIDENTIALITE et TYPE sont les mêmes que celles énoncées pour le premier groupe d'objets : ENV-RANG, ARMOIRE, TIROIR, ...

12. STATUT-DE-CONFIDENTIALITE :

13. TYPE :

14. POSSIBILITE-D'ACCES-CONSULTATION :

. Signification : l'attribut POSSIBILITE-D'ACCES-CONSULTATION d'un objet permet à son propriétaire de fixer la liste des groupes d'utilisateurs qui peuvent y accéder en consultation.

Cette liste de groupes d'utilisateurs reprendra les noms de ces groupes.

. Propriétés :

- NON IDENTIFIANT
- REPETITIF
- ELEMENTAIRE
- OBLIGATOIRE
- MODIFIABLE
- MANUEL
- TYPE DE VALEUR : chaîne finie de caractères alphanumériques.
  exemples : "Groupe-bureautique", "Groupe-O.S.", ...

15. POSSIBILITE-D'ACCES-MISE-A-JOUR :

. Signification : l'attribut POSSIBILITE-D'ACCES-MISE-A-JOUR d'un objet permet à propriétaire de fixer la liste des groupes d'utilisateurs qui peuvent avoir y accéder en mise à jour.

Cette liste de groupes d'utilisateurs reprendra les noms de ces groupes.

. Propriétés :

- NON IDENTIFIANT
- REPETITIF
- ELEMENTAIRE
- OBLIGATOIRE
- MODIFIABLE
- MANUEL
- TYPE DE VALEUR : chaîne finie de caractères alphanumériques.
  exemples : "Groupe-direction", "Groupe-comptabilité", ...

16. EXPEDITION :

. Signification : cet attribut a pour objectif de reprendre certains renseignements concernant l'éventuelle expédition de cet objet par courrier électronique par exemple. Le propriétaire de l'objet peut mettre à jour ces renseignements.

Exemples de renseignements pouvant être liés à l'expédition :

- *le nom de l'expéditeur ;*
- *le nom du ou des destinataires ;*
- *la date d'expédition ;*
- *la date de réception ;*
- *s'il y a attente de réponse ;*
- *s'il y a attente de confirmation de réception ;*
- *la liste des réponses.*

Cet attribut ne s'applique qu'au T.E. INFORMATION.

. *Propriétés :*

- *NON IDENTIFIANT*
- *SIMPLE*
- *ELEMENTAIRE*
- *FACULTATIF*
- *MODIFIABLE*
- *MANUEL*
- *TYPE DE VALEUR : chaîne finie de caractères alphanumériques.*

exemple :
" - expéditeur : Stefan ROBERT
- destinataires : R. LESUISSE
- date d'expédition : 03-12-85
- date de réception : ?
- attente de réponse : OUI
- liste des réponses : / "

17. MOTS-CLES :

. *Signification : cet attribut permet, comme l'attribut DESCRIPTION, de décrire l'objet mais de manière plus formelle sur base de mots-clés. On pourra par exemple demander tous les objets qui reprennent les mots-clés : X, Y et Z.*

Cet attribut ne s'applique qu'au T.E. INFORMATION.

. *Propriétés :*

- *NON IDENTIFIANT*
- *REPETITIF*
- *ELEMENTAIRE*
- *FACULTATIF*
- *MODIFIABLE*
- *MANUEL*
- *TYPE DE VALEUR : chaîne finie de caractères alphanumériques.*

exemples : "INTERFACES H-M", "ALGORITHMES", "CAO".

18. REFERENCE-STOCKAGE :

. Signification : cet attribut représente une référence à la localisation de l'objet sur support informatique. Il s'agit en fait du nom du fichier dans lequel se trouve le contenu de l'objet.

Cet attribut ne s'applique qu'au T.E. INFORMATION.

. Propriétés :

- NON IDENTIFIANT
- SIMPLE
- ELEMENTAIRE
- FACULTATIF
- MODIFIABLE
- MANUEL
- TYPE DE VALEUR : chaîne finie de caractères alphanumériques.
  exemple : "/usr/etd/robert/rapport.doc".

a. Facultatif :

Les objets gérés par l'E.R. ne doivent pas nécessairement être sur support informatique.

19. REFERENCE-ORIGINAL :

. Signification : cet attribut représente une référence à l'original de l'objet. Cette propriété est exigée étant donné que pour des raisons juridiques, il est obligatoire de garder l'original de certains documents enregistrés sur support informatique. En cas de litige, seul lui aura force de preuve.

D'autre part, l'objet pourrait ne pas être sur support informatique, comme c'est le cas d'un grand nombre d'informations. La référence pourrait être, par exemple, la cote d'un ouvrage ou d'un article d'une revue dans une bibliothèque.

Cet attribut ne s'applique qu'au T.E. INFORMATION.

. Propriétés :

- NON IDENTIFIANT
- SIMPLE
- ELEMENTAIRE
- FACULTATIF
- MODIFIABLE
- MANUEL
- TYPE DE VALEUR : chaîne finie de caractères alphanumériques.
  exemple : "I535/27/1".

F. Attributs des T.E. du groupe : UTILISATEUR et GROUPE-UTILISATEUR :

Dans cette partie, nous utiliserons également le terme "objet" pour UTILISATEUR et GROUPE-UTILISATEUR.

Liste des attributs du groupe :

- NOM
- CODE
- MOT-DE-PASSE
- DATE-DE-CREATION
- CREATEUR
- PERMISSION-D'ACCES

1. NOM :

. Signification : l'attribut NOM d'un objet permettra d'identifier cet objet parmi les autres objets de son type. Il permettra également au système d'accéder aux objets de ce type.

. Propriétés :

- IDENTIFIANT LOCAL
- SIMPLE
- ELEMENTAIRE
- OBLIGATOIRE
- MODIFIABLE
- MANUEL
- TYPE DE VALEUR : chaîne finie de caractères alphanumériques.
  exemples : "Jean", "Groupe-Bureautique"

a. Modifiable

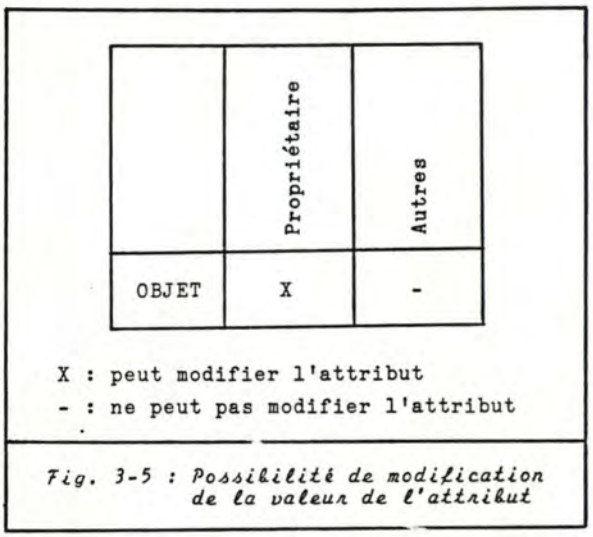
Seul l'administrateur de l'E.R. pourra créer des utilisateurs et des groupes d'utilisateurs. Il sera donc le seul à pouvoir changer leur nom.

On ne peut donc distinguer que deux classes d'utilisateurs :

. le propriétaire de l'objet ;

. les autres utilisateurs.

La figure 3-5 illustre la possibilité de modification qui servira de référence à tous les attributs modifiables de ce présent groupe d'objets.



	Propriétaire	Autres
OBJET	X	-

X : peut modifier l'attribut
- : ne peut pas modifier l'attribut

Fig. 3-5 : Possibilité de modification de la valeur de l'attribut

2. *CODE* :

. *Signification* : *le CODE est un identificateur interne qui permettra au système de gérer les objets.*

. *Propriétés* :

- *IDENTIFIANT GLOBAL*
- *SIMPLE*
- *ELEMENTAIRE*
- *OBLIGATOIRE*
- *DEFINITIF*
- *AUTOMATIQUE*
- *TYPE DE VALEUR : nombre entier.*
  exemples : 00023, 12478, 00007.

Les remarques sont les mêmes que celles qui ont été données pour les objets du premier groupe.

3. *MOT-DE-PASSE* :

. *Signification* : *le mot de passe de l'utilisateur est un code secret qui lui permet de se protéger.*

Cet attribut ne s'applique qu'au T.E. UTILISATEUR.

. *Propriétés* :

- *IDENTIFIANT LOCAL*
- *SIMPLE*
- *ELEMENTAIRE*

- OBLIGATOIRE
- MODIFIABLE
- MANUEL
- TYPE DE VALEUR : chaîne finie de caractères alphanumériques.
  exemple : "CODE007"

4. DATE-DE-CREATION :

La signification et les propriétés des attributs DATE-DE-CREATION et CREATEUR sont les mêmes que celles énoncées pour le premier groupe d'objets : ENV-RANG, ARMOIRE, TIROIR, FARDE, POUBELLE et BOITE.

5. CREATEUR :

6. PERMISSION-D'ACCES :

. Signification : il s'agit d'un attribut indiquant si cet utilisateur peut modifier des objets dans l'E.R. (création ou mise à jour) ou s'il est simplement autorisé à consulter ceux-ci.

Cet attribut ne s'applique qu'au T.E. UTILISATEUR.

. Propriétés :

- NON IDENTIFIANT
- SIMPLE
- ELEMENTAIRE
- OBLIGATOIRE
- MODIFIABLE
- MANUEL
- TYPE DE VALEUR : (ECRITURE, LECTURE).

2.2.3. Les associations

La figure 3-6 rappelle le schéma E-A illustré à la figure 3-1 en faisant ressortir les associations.

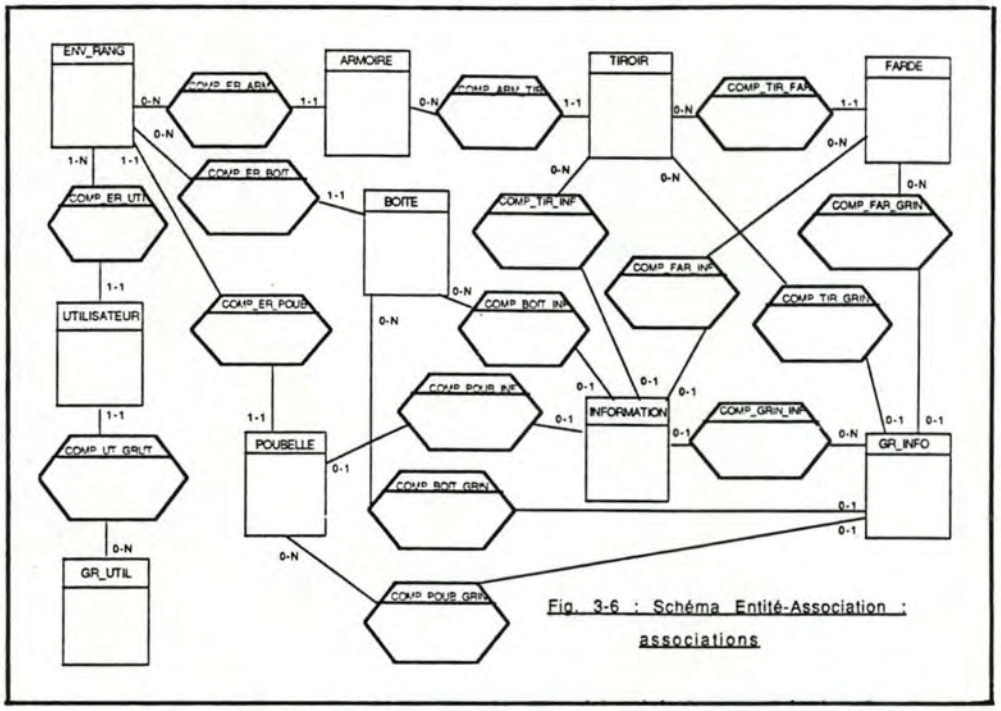


Fig. 3-6 : Schéma Entité-Association : associations

Dans le schéma E-A (figure 3-6), les types d'association (T.A.) sont représentés graphiquement par des hexagones (figure 3-7). Tous les T.A. y sont de degré 2 (binaires) parce qu'ils relient deux T.E.

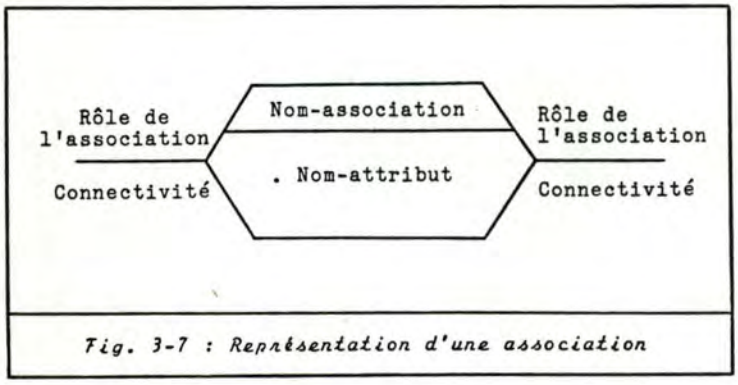


Fig. 3-7 : Représentation d'une association

La signification des T.A. est identique pour tous, aux T.E. près qu'ils relient comme l'illustre de manière générique la figure 3-8. L'interprétation en est la suivante :

- une occurrence du T.E. "TE1" "est composée" d'occurrence(s) du T.E. "TE2" ;

- une occurrence du T.E. "TE2" "fait partie" d'une occurrence du T.E. "TE1".

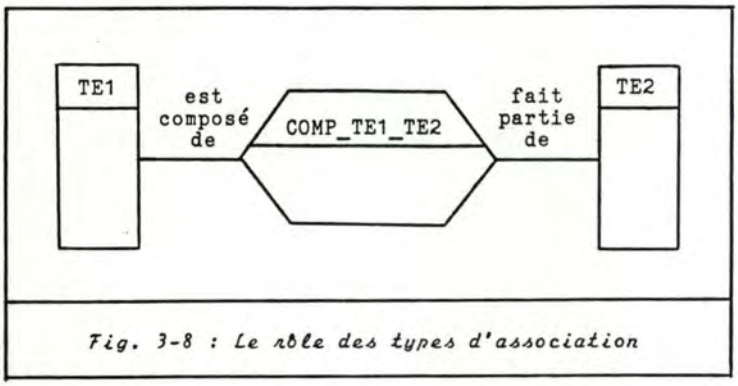


Fig. 3-8 : Le rôle des types d'association

A. Liste des associations :

. COMPOSITION-ER-ARMOIRE
. COMPOSITION-ER-BOITE
. COMPOSITION-ER-POUBELLE
. COMPOSITION-ER-UTILISATEUR
. COMPOSITION-ARMOIRE-TIROIR
. COMPOSITION-BOITE-GROUPE-INFORMATION
. COMPOSITION-BOITE-INFORMATION
. COMPOSITION-POUBELLE-GROUPE-INFORMATION
. COMPOSITION-POUBELLE-INFORMATION
. COMPOSITION-TIROIR-GROUPE-INFORMATION
. COMPOSITION-TIROIR-FARDE
. COMPOSITION-TIROIR-INFORMATION
. COMPOSITION-FARDE-GROUPE-INFORMATION
. COMPOSITION-FARDE-INFORMATION
. COMPOSITION-GROUPE-INFORMATION-INFORMATION
. COMPOSITION-UTILISATEUR-GROUPE-UTILISATEUR

2.2.3.2. Interprétation

COMP-ER-ARM :

Nom : COMPOSITION-ER-ARMOIRE
Association reliant les entités : E.R. et ARMOIRE.
A un E.R. peut correspondre 0,1 ou plusieurs ARMOIRES.
A une ARMOIRE correspond 1 et 1 seul E.R.

COMP-ER-BOIT :

Nom : COMPOSITION-ER-BOITE.
Association reliant les entités : E.R. et BOITE.
A un E.R. peut correspondre 0,1 ou plusieurs BOITES.
A une BOITE correspond 1 et 1 seul E.R.

COMP-ER-POUB :

Nom : COMPOSITION-ER-POUBELLE.
Association reliant les entités : E.R. et POUBELLE.
A un E.R. ne correspond qu'une seule POUBELLE .
A une POUBELLE correspond 1 et 1 seul E.R.

COMP-ER-UTI :

Nom : COMPOSITION-ER-UTILISATEUR.
Association reliant les entités : E.R. et UTILISATEUR.
A un E.R. peut correspondre 0,1 ou plusieurs UTILISATEURS.
A un UTILISATEUR correspond 1 et 1 seul E.R.

COMP-ARM-TIR :

Nom : COMPOSITION-ARMOIRE-TIROIR.
Association reliant les entités : ARMOIRE et TIROIR.
A une ARMOIRE peut correspondre 0,1 ou plusieurs TIROIRS.
A un TIROIR correspond 1 et 1 seul ARMOIRE .

COMP-BOIT-GRIN :

Nom : COMPOSITION-BOITE-GROUPE-INFORMATION.
Association reliant les entités : BOITE et GROUPES-INFORMATION.
A une BOITE peut correspondre 0,1 ou plusieurs GROUPE-INFORMATION.
A un GROUPE-INFORMATION correspond au plus une BOITE.

COMP-BOIT-INF :

Nom : COMPOSITION-BOITE-INFORMATION.
Association reliant les entités : BOITE et INFORMATION.
A une BOITE peut correspondre 0,1 ou plusieurs INFORMATIONS.
A une INFORMATION correspond au plus une BOITE.

COMP-POUB-GRIN :

Nom : COMPOSITION-POUBELLE-GROUPE-INFORMATION.
Association reliant les entités : POUBELLE et GROUPE-INFORMATION.
A une POUBELLE peut correspondre 0,1 ou plusieurs GROUPES-INFORMATION.
A un GROUPE-INFORMATION correspond au plus une POUBELLE.

COMP-POUB-INF :

Nom : COMPOSITION-POUBELLE-INFORMATION.
Association reliant les entités : POUBELLE et INFORMATION.
A une POUBELLE peut correspondre 0,1 ou plusieurs INFORMATIONS.
A une INFORMATION correspond au plus une POUBELLE.

COMP-TIR-GRIN :

Nom : COMPOSITION-TIROIR-GROUPE-INFORMATION.
Association reliant les entités : TIROIR et GROUPE-INFORMATION.
A un TIROIR peut correspondre 0,1 ou plusieurs GROUPES-INFORMATION.
A un GROUPE-INFORMATION correspond au plus un TIROIR.

COMP-TIR-FAR :

Nom : COMPOSITION-TIROIR-FARDE.
Association reliant les entités : TIROIR et FARDE.
A un TIROIR peut correspondre 0,1 ou plusieurs FARDES.
A une FARDE correspond 1 et 1 seul TIROIR.

COMP-TIR-INF :

Nom : COMPOSITION-TIROIR-INFORMATION.
Association reliant les entités : TIROIR et INFORMATION.
A un TIROIR peut correspondre 0,1 ou plusieurs INFORMATIONS.
A une INFORMATION correspond au plus un TIROIR.

COMP-FAR-GRIN :

Nom : COMPOSITION-FARDE-GROUPE-INFORMATION.
Association reliant les entités : FARDE et GROUPE-INFORMATION.
A une FARDE peut correspondre 0,1 ou plusieurs GROUPES-INFORMATION.
A un GROUPE-INFORMATION correspond au plus une FARDE.

COMP-FAR-INF :

Nom : COMPOSITION-FARDE-INFORMATION.
Association reliant les entités : FARDE et INFORMATION.
A une FARDE peut correspondre 0,1 ou plusieurs INFORMATIONS.
A une INFORMATION correspond au plus une FARDE.

COMP-GRIN-INF :

Nom : COMPOSITION-GROUPE-INFORMATION-INFORMATION.
Association reliant les entités : GROUPE-INFORMATION et INFORMATION.
A un GROUPE-INFORMATION peut correspondre 0,1 ou plusieurs INFORMATIONS.
A une INFORMATION correspond au plus un GROUPE-INFORMATION.

COMP-UT-GRUT :

Nom : COMPOSITION-UTILISATEUR-GROUPE-UTILISATEUR.
Association reliant les entités : UTILISATEUR et GROUPE-UTILISATEUR.
A un GROUPE-UTILISATEUR peut correspondre 0,1 ou plusieurs UTILISATEURS.
A une UTILISATEUR correspond 1 et 1 seul GROUPE-UTILISATEUR.

2.2.4. Les contraintes d'intégrité du schéma E-A

Définition : "une contrainte d'intégrité (C.I.) est une propriété non représentée par les concepts de base du modèle que doivent satisfaire les informations appartenant à la mémoire du système d'information (S.I.)" (BOPI83).

2.2.4.1. Contraintes d'intégrité portant sur les entités

A. Contraintes d'existence :

1. Une occurrence de UTILISATEUR ne peut exister que si elle participe à une COMPOSITION-ER-UTILISATEUR et à une COMPOSITION-UTILISATEUR-GROUPE-UTILISATEUR.

2. Une occurrence de POUBELLE ne peut exister que si elle participe à une COMPOSITION-ER-POUBELLE.

3. Une occurrence de BOITE ne peut exister que si elle participe à une COMPOSITION-ER-BOITE.

4. Une occurrence d' ARMOIRE ne peut exister que si elle participe à une COMPOSITION-ER-ARMOIRE.

5. Une occurrence de TIROIR ne peut exister que si elle participe à une COMPOSITION-ARMOIRE-TIROIR.

6. Une occurrence de FARDE ne peut exister que si elle participe à une COMPOSITION-TIROIR-FARDE.

B. Identifiant d'un type d'entité :

Les identifiants des T.E. ont été analysés dans les propriétés des attributs (cfr. 2.2.2. C. Définition des propriétés des attributs).

2.2.4.2. Contraintes d'intégrité portant sur les associations

A. Contraintes d'exclusion

Définition : "Il y a contrainte d'exclusion lorsque la participation d'un T.E. dans une occurrence d'un type d'association (T.A.) exclut sa participation dans une occurrence d'un autre T.A." (BOPI83).

1. Une occurrence d' INFORMATION doit participer exclusivement à une occurrence de :

   COMPOSITION-INFORMATION-GROUPE-INFORMATION
   ou COMPOSITION-FARDE-INFORMATION
   ou COMPOSITION-TIROIR-INFORMATION
   ou COMPOSITION-BOITE-INFORMATION
   ou COMPOSITION-POUBELLE-INFORMATION

2. Une occurrence de GROUPE-INFORMATION doit participer exclusivement à une occurrence de :

COMPOSITION-FARDE-INFORMATION
ou COMPOSITION-TIROIR-INFORMATION
ou COMPOSITION-BOITE-INFORMATION
ou COMPOSITION-POUBELLE-INFORMATION

B. Identifiant d'une association

Définition : "une association d'un type donné sera identifiée par les identifiants des entités sur lesquelles elle est définie. En effet, l'existence d'une association est contingente à l'existence des entités qu'elle met en correspondance" (BOPI83).

C. Autres

Nous n'avons pas respecté les règles du modèle E-A comme décrit dans (BOPI83) pour la représentation de certaines associations. Ceci est dû à la complexité que cela aurait entraîné dans la représentation graphique du schéma E-A.

Ces associations ont été remplacées par des attributs d'entités. Il s'agit des attributs :

- POSSIBILITE-D'ACCES
- POSSIBILITE-D'ACCES-CONSULTATION
- POSSIBILITE-D'ACCES-MISE-A-JOUR
- CREATEUR
- DATE-DE-CREATION
- MODIFICAT
- DATE-DE-MISE-A-JOUR
- CONSULT
- DATE-DE-CONSULTATION

1. *Les attributs POSSIBILITE-D'ACCES, POSSIBILITE-D'ACCES-CONSULTATION, et POSSIBILITE-D'ACCES-MISE-A-JOUR correspondent en fait à une association entre le T.E. auquel appartient ces attributs et le T.E. GROUPE-UTILISATEUR.*

   La figure 3-9 illustre l'attribut POSSIBILITE-D'ACCES pour les T.E. : ENVIRONNEMENT-RANGEMENT, POUBELLE, BOITE, ARMOIRE, TIROIR et FARDE (*).

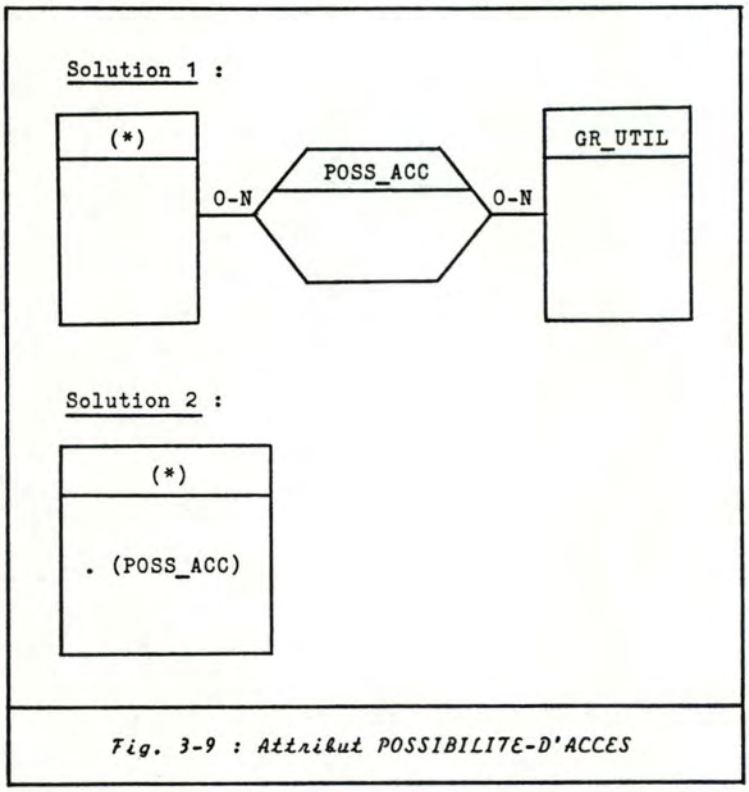


Fig. 3-9 : Attribut POSSIBILITE-D'ACCES

La figure 3-10 illustre les attributs POSSIBILITE-D'ACCES-CONSULTATION et POSSIBILITE-D'ACCES-MISE-A-JOUR pour les T.E. : INFORMATION et GROUPE-INFORMATION (*).

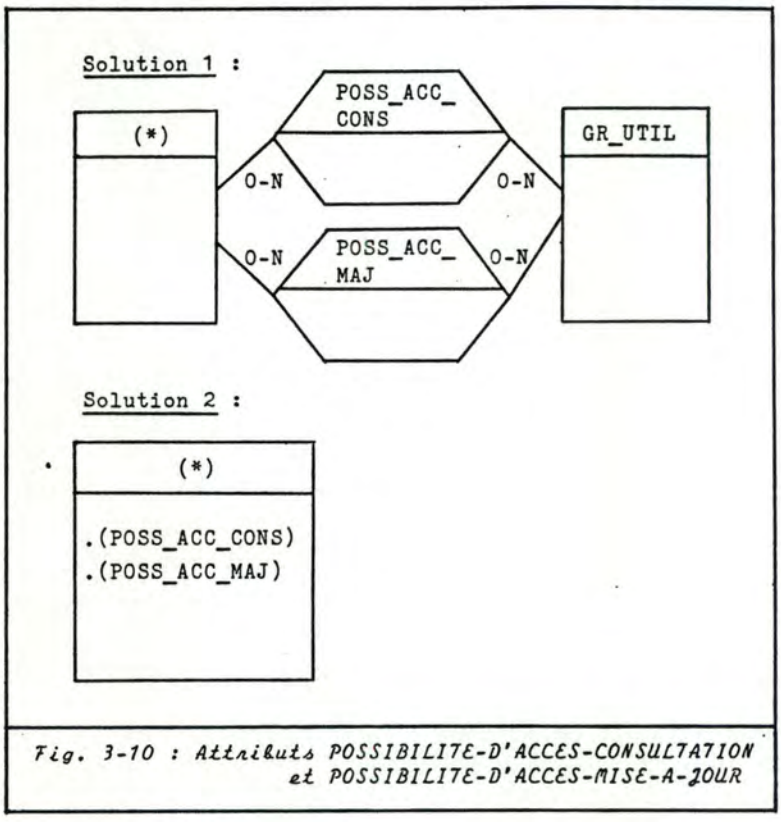


Fig. 3-10 : Attributs POSSIBILITE-D'ACCES-CONSULTATION et POSSIBILITE-D'ACCES-MISE-A-JOUR

Dans les figures 3-9 et 3-10, nous avons opté pour la deuxième solution alors que la solution correcte est la première. Nous devons donc compléter notre solution par une contrainte sur la valeur des attributs POSSIBILITE-D'ACCES, POSSIBILITE-D'ACCES-CONSULTATION, et POSSIBILITE-D'ACCES-MISE-A-JOUR. Ces attributs ne pourront prendre comme valeur que des valeurs existantes de l'attribut nom d'occurrences du T.E. GROUPE-UTILISATEUR.

2. *Les attributs CREATEUR, CONSULT et MODIFICAT correspondent en fait à une association entre le T.E. auquel appartient ces attributs et le T.E. UTILISATEUR.*

La figure 3-11 illustre les attributs CREATEUR et DATE-DE-CREATION pour les T.E. : UTILISATEUR, GROUPE-UTILISATEUR, ENVIRONNEMENT-RANGEMENT, POUBELLE, BOITE, ARMOIRE, TIROIR, FARDE et GROUPE-INFORMATION.

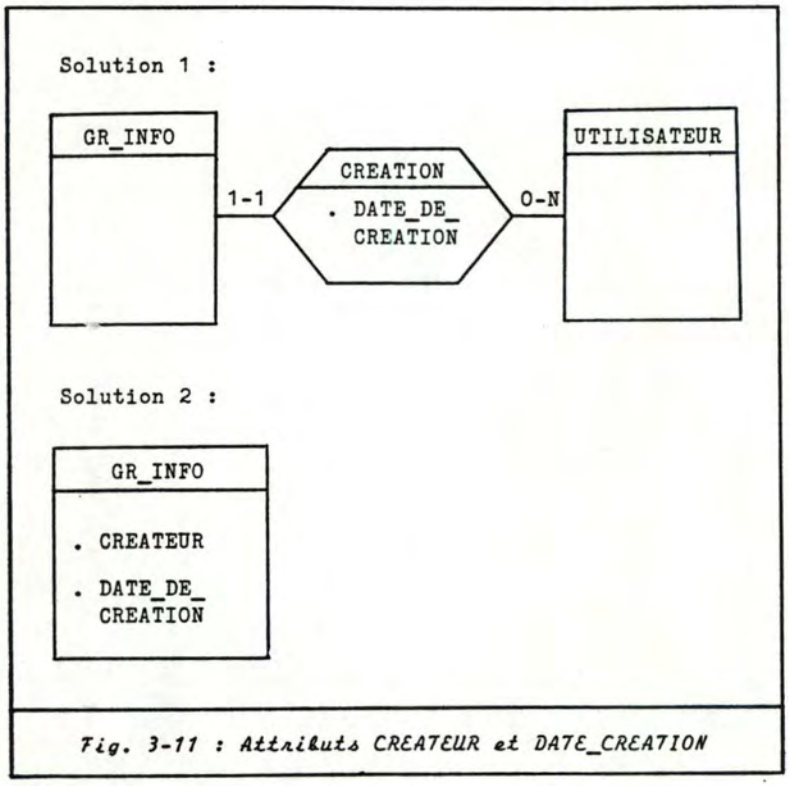


Fig. 3-11 : Attributs CREATEUR et DATE_CREATION

La figure 3-12 illustre les attributs CREATEUR, DATE-DE-CREATION, CONSULT, DATE-DE-CONSULTATION, MODIFICAT, DATE-DE-MISE-A-JOUR pour le T.E. : INFORMATION.

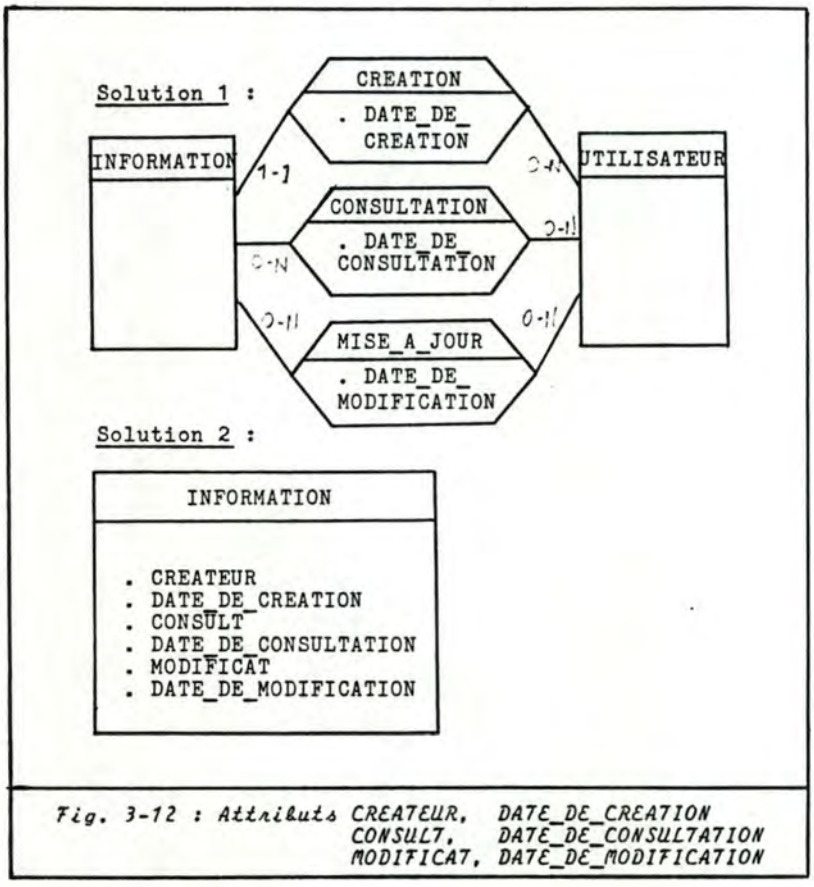


Fig. 3-12 : Attributs CREATEUR, DATE_DE_CREATION CONSULT, DATE_DE_CONSULTATION MODIFICAT, DATE_DE_MODIFICATION

Dans les figures 3-11 et 3-12, nous avons opté pour la deuxième solution alors que la solution correcte est la première. Nous devons donc compléter notre solution par une contrainte sur la valeur des attributs CREATEUR, DATE-DE-CREATION, CONSULT, DATE-DE-CONSULTATION, MODIFICAT, DATE-DE-MISE-A-JOUR. Ces attributs ne pourront prendre comme valeur que des valeurs existantes de l'attribut code d'occurrences du T.E. UTILISATEUR.

Remarque :

Notons que les attributs DATE-DE-CONSULTATION et DATE-DE-MISE-A-JOUR n'ont pas la même sémantique dans les deux solutions présentées à la figure 3-12. La première solution permet d'avoir un historique des consultations et des mises à jour des informations, alors que la deuxième solution ne le permet pas. Cette solution ne peut conserver qu'un couple de dates par entité. Nous conviendrons qu'il s'agit respectivement

de la dernière date de consultation et de la dernière date de modification de cette entité.

Cette remarque ne concerne pas l'attribut DATE-DE-CREATION dans la mesure où l'on considère comme trivial qu'un objet ne peut être créé qu'une seule fois.

2.2.4.3. Contraintes d'intégrité portant sur les attributs

A. Contraintes de valeur :

1. Les contraintes de valeurs définissent l'ensemble des valeurs que peut prendre un attribut. Elles ont déjà été abordées dans les propriétés des attributs (cfr. type de valeur dans 2.2.2. C. Définition des propriétés des attributs) ;

2. D'autres contraintes peuvent définir les valeurs que peut prendre un attribut en fonction des valeurs prises par d'autres :

   a) Les valeurs de DATE-DE-CONSULTATION et DATE-DE-MISE-A-JOUR doivent être supérieures ou égales à la valeur de DATE-DE-CREATION de l'information à laquelle ils appartiennent. Ceci parce que la valeur de l'attribut DATE-DE-CREATION ne peut être modifiée et que les attributs DATE-DE-CONSULTATION et DATE-DE-MISE-A-JOUR prennent la valeur de DATE-DE-CREATION lors de la création de l'information dans l'E.R.

   b) La possibilité d'une mise à jour d'une information étant plus restrictive que celle d'une simple consultation, la participation dans la liste POSSIBILITE-D'ACCES-MISE-A-JOUR implique automatiquement la participation dans la liste POSSIBILITE-D'ACCES-CONSULTATION sans que cela doive y figurer explicitement.

   c) Les groupes d'utilisateurs de la liste POSSIBILITE-D'ACCES d'une occurrence de TIROIR doivent être inclus dans la liste POSSIBILITE-D'ACCES de l'occurrence ARMOIRE à laquelle ce tiroir est associé. Et ceci pour être cohérent dans la méthode d'accès.

   d) Les groupes d'utilisateurs de la liste POSSIBILITE-D'ACCES d'une occurrence de FARDE doivent être inclus dans la liste POSSIBILITE-D'ACCES de l'occurrence ARMOIRE à laquelle ce tiroir est associé.

   e) Les groupes d'utilisateurs de la liste POSSIBILITE-D'ACCES d'une occurrence de ARMOIRE, de BOITE ou de POUBELLE doivent être inclus dans la liste POSSIBILITE-D'ACCES de l'occurrence ENVIRONNEMENT-RANGEMENT. Ceci est toujours réalisé puisque nous avons dit que l'accès à l'E.R. serait accordé à tous les groupes d'utilisateurs. La liste

POSSIBILITE-D'ACCES de l'E.R. se compose automatiquement de tous les groupes d'utilisateurs.

f) Les groupes d'utilisateurs de la liste POSSIBILITE-D'ACCES-CONSULTATION et POSSIBILITE-D'ACCES-MISE-A-JOUR d'une occurrence d'INFORMATION doivent faire partie de la liste POSSIBILITE-D'ACCES de l'entité à laquelle cette information est associée. Cette entité peut être soit :

un GROUPE-INFORMATION
ou une FARDE
ou un TIROIR
ou une BOITE
ou la POUBELLE

g) Les groupes d'utilisateurs de la liste POSSIBILITE-D'ACCES-CONSULTATION et POSSIBILITE-D'ACCES-MISE-A-JOUR d'une occurrence de GROUPE-INFORMATION doivent être inclus dans la liste POSSIBILITE-D'ACCES de l'entité à laquelle ce groupe-information est associé. Cette entité peut être soit :

une FARDE
ou un TIROIR
ou une BOITE
ou la POUBELLE

2.3. AUTRES DONNEES

Pour que l'E.R. puisse être géré, deux données supplémentaires seront utilisées :

1. *Date du jour : la date du jour peut être une date que l'utilisateur introduit avant l'appel d'une opération sur un objet de l'E.R. Elle est nécessaire, par exemples, pour mémoriser la date de création, consultation ou mise à jour d'une information dans l'E.R.*

2. *Compteur de codes : un compteur est nécessaire pour la génération automatique des codes des nouveaux objets. Il est incrémenté de une unité à chaque création d'un objet. Chaque objet reçoit ainsi un code distinct qui servira d'identifiant global de l'objet.*

3. SPECIFICATION DES OPERATIONS SUR LES OBJETS DE L'E.R.

La présente section se compose de deux parties :

- *une première partie a pour but de dresser la portée des opérations dans le cadre de ce qui a été implémenté pour ce mémoire ;*

- *une deuxième partie analyse de manière détaillée les différentes opérations en termes des pré- et postconditions, des données en entrée, des résultats en sortie et de l'objectif proprement dit de l'opération.*

3.1. PORTEE DES OPERATIONS

On se limitera, dans le cadre de ce mémoire, à spécifier et à implémenter une couche de primitives qui permettront de gérer les objets de l'E.R. Cette couche d'opérations et l'E.R., représenté par le symbole d'une base de données, sont illustrés à la figure 3-13.

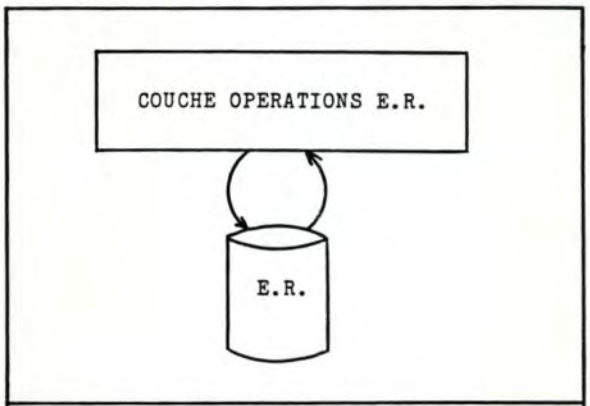


Fig. 3-13 : Couche opérations E.R.

Cette couche d'opérations a pour objectif de servir de base à une application future. Cette application pourra être, par exemple, un courrier électronique, un service intégré de rangemement de documents, ... Elle fera appel à l'E.R. pour classer les informations. La figure 3-14 illustre l'E.R. et les opérations vus de l'application future..

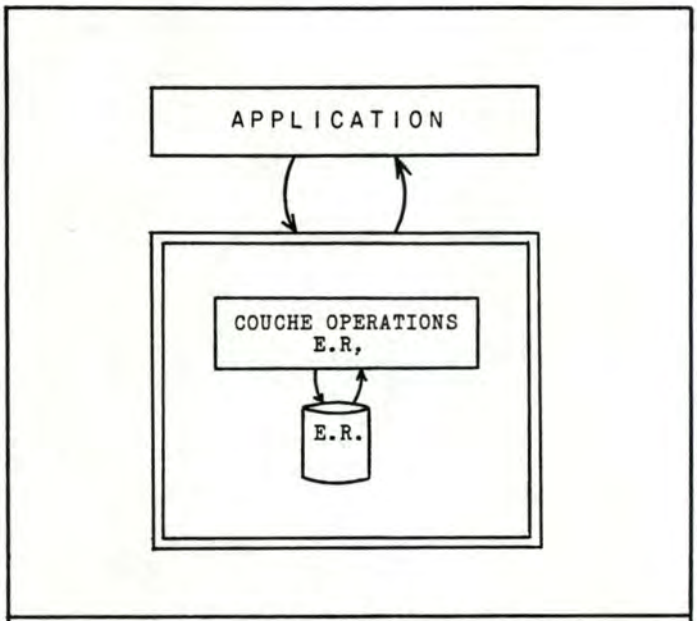


Fig. 3-14 : Application

Rappelons qu'aucune opération de cette couche ne travaille sur le contenu des informations. Elles s'occupent essentiellement de la fonction de rangement et donc des propriétés des objets et non de le leur valeur. Le seul lien entre un objet information et l'information proprement dite est la référence qui pourra, par exemple, être la référence d'un fichier ou d'une localisation dans une bibliothèque. Celle-ci permet à l'E.R. de gérer des informations sur support informatique mais aussi des informations sur d'autres supports tels que le papier ou le microfilm.

Enfin, cette couche d'opérations s'adressant à une application et donc à des concepteurs et programmeurs, ces opérations seront spécifiées à la manière des fonctions, c'est-à-dire en terme :

- *des données en entrée ;*
- *des préconditions à respecter avant l'appel de la fonction ;*
- *de l'objectif de la fonction ;*
- *des résultats en sortie ;*
- *des postconditions à la sortie de la fonction.*

3.2. LES OPERATIONS

A. Liste des opérations :

- CREATION
- SUPPRESSION
- MODIFICATION
- CONSULTATION
- MISE A JOUR
- RECHERCHE
- PROPRIETE

Particularité : pour limiter le nombre d'opérations, et ainsi le nombre de points d'entrée dans la couche des opérations, nous n'avons pas distingué les opérations par type d'objet. Il n'existe donc qu'une opération CREATION, qui servira aussi bien à la création d'une armoire, d'une information ou d'un nouvel utilisateur. Cette distinction se fera cependant à l'intérieur de chaque fonction et notamment au niveau de certaines conditions à respecter qui sont différentes en fonction des types d'objet.

B. Spécifications

1. CREATION (utilisateur, date jour, objet, code-retour)

a) Données :

- le code, la permission et le groupe de l'utilisateur ;
- la date du jour ;
- le type et l'ensemble des attributs manuels (1) de l'objet à créer.

b) Préconditions :

- la base de données de l'E.R. doit être ouverte (2) ;

---

(1) Le terme manuel est un type de valeur que l'on a utilisé lors de l'analyse fonctionnelle des données, au point 2.2.2. Les attributs des T.E., de ce présent chapitre.

(2) Pour une question de performance de temps d'accès, il est préférable que la base de données (BD) soit ouverte avant l'appel de la fonction ; ceci pour éviter de devoir ouvrir puis refermer la BD à chaque appel de fonction. A ce propos, les temps d'accès des différentes opérations sont donnés, à titre illustratif dans le cinquième chapitre : 5. Performance de l'implémentation.

- les données fournies en entrée doivent être valides.

c) Objectif de la fonction :

La fonction création a pour objectif de permettre la création, à la demande d'un utilisateur, d'un nouvel objet dans l'E.R.

Conditions à remplir :

- L'utilisateur doit avoir la permission 'ECRITURE'.

- Pour les objets de type UTILISATEUR, GROUPE-UTILISATEUR, BOITE, ARMOIRE et TIROIR : l'utilisateur doit être l'administrateur de l'E.R.

- Pour les objets de type ENV-RANG et POUBELLE, la création n'est pas autorisée.

- Pour les objets de type FARDE, INFORMATION et GROUPE-INFORMATION :

  . le contenant (3) de l'objet à créer doit exister ;

  . l'utilisateur doit avoir accès au contenant.

d) Résultats :

- code-retour : un code d'erreur est renvoyé au retour de chaque appel de la fonction. Une valeur égale à 0 signifie que l'opération s'est bien passée. Sinon, il y a eu une erreur (4) ;

- les attributs d'objet complétés ;

- la création de l'objet dans l'E.R. si l'opération s'est bien déroulée.

e) Postcondition :

- Si le code-retour est égal à 0, l'objet sera ajouté a l'E.R. Sinon, l'E.R. sera dans l'état précédant l'appel à la fonction.

---

(3) Le terme contenant est utilisé pour désigner un objet qui en contient un autre.

(4) La liste des codes d'erreur, par type, est reprise à l'annexe A, fichier LIB-C.QC.

2. SUPPRESSION (utilisateur, date jour, objet, code-retour)

a) Données :

- le code, la permission et le groupe de l'utilisateur ;
- la date du jour ;
- le type et le code de l'objet à supprimer.

b) Préconditions :

- la base de données de l'E.R. doit être ouverte ;
- les données fournies en entrée doivent être valides.

c) Objectif de la fonction :

La fonction suppression a pour objectif de permettre la suppression, à la demande d'un utilisateur, d'un objet dans l'E.R.

Conditions à remplir :

- Pour les objets de type BOITE, ARMOIRE et TIROIR : l'utilisateur doit être l'administrateur de l'E.R.
- Pour les objets de type UTILISATEUR, GROUPE-UTILISATEUR, ENV-RANG et POUBELLE, la suppression n'est pas autorisée.
- Pour les objets de type FARDE, INFORMATION et GROUPE-INFORMATION : l'utilisateur doit être l'administrateur ou le proriétaire (CREATEUR) de l'objet.
- Pour les objets de type BOITE, ARMOIRE, TIROIR, FARDE, et GROUPE-INFORMATION : l'objet doit être vide, il ne peut plus contenir aucun objet.
- Pour les objets de type INFORMATION : l'objet doit être présent dans l'E.R., c'est-à-dire ni en consultation, ni en mise à jour.

d) Résultats :

- code-retour : un code d'erreur est renvoyé au retour de chaque appel de la fonction. Une valeur égale à 0 signifie que l'opération s'est bien passée. Sinon, il y a eu une erreur (4) ;
- la suppression de l'objet de l'E.R. si l'opération s'est bien déroulée.

e) Postcondition :

- Si le code-retour est égal à 0, l'objet sera supprimé de l'E.R. Sinon, l'E.R. sera dans l'état précédant l'appel à la fonction.

3. MODIFICATION (utilisateur, date jour, objet, code-retour)

a) Données :

- le code, la permission et le groupe de l'utilisateur ;

- la date du jour ;

- le type et l'ensemble des nouveaux attributs modifiables (5) de l'objet à modifier.

b) Préconditions :

- la base de données de l'E.R. doit être ouverte ;

- les données fournies en entrée doivent être valides.

c) Objectif de la fonction :

La fonction modification a pour objectif de permettre la modification, à la demande d'un utilisateur, d'un ou plusieurs attributs d'un objet de l'E.R.

Conditions à remplir :

- L'utilisateur doit avoir la permission 'ECRITURE'.

- Pour les objets de type : UTILISATEUR, GROUPE-UTILISATEUR, ENV-RANG et POUBELLE, BOITE, ARMOIRE et TIROIR, l'utilisateur doit être l'administrateur de l'E.R.

- Pour les objets de type FARDE, INFORMATION et GROUPE-INFORMATION : l'utilisateur doit être l'administrateur de l'E.R ou le proriétaire (CREATEUR) de l'objet.

d) Résultats :

- code-retour : un code d'erreur est renvoyé au retour de chaque appel de la fonction. Une valeur égale à 0 signifie que l'opération s'est bien passée. Sinon, il y a eu une erreur (4) ;

---

(5) Le terme modifiable est un type de valeur que l'on a utilisé lors de l'analyse fonctionnelle des données, au point 2.2.2. Les attributs des T.E., de ce présent chapitre.

- la modification des attributs de l'objet de l'E.R. si l'opération s'est bien déroulée.

e) Postcondition :

- Si le code-retour est égal à 0, l'objet sera modifié de l'E.R. Sinon, l'E.R. sera dans l'état précédant l'appel à la fonction.

4. CONSULTATION (utilisateur, date jour, objet, mode, code-retour)

a) Données :

- le code, la permission et le groupe de l'utilisateur ;

- la date du jour ;

- le mode d'ouverture (OUVERTURE ou FERMETURE) ;

- le type et le code de l'objet à consulter.

b) Préconditions :

- la base de données de l'E.R. doit être ouverte ;

- les données fournies en entrée doivent être valides.

c) Objectif de la fonction :

La fonction consultation a pour objectif de permettre la consultation, à la demande d'un utilisateur, d'une information ou d'un groupe d'informations.

Conditions à remplir :

- L'utilisateur doit avoir accès à l'objet en consultation ;

- Si l'information est en mode CONFIDENTIEL, seul le propriétaire de celle-ci peut la consulter.

d) Résultats :

- code-retour : un code d'erreur est renvoyé au retour de chaque appel de la fonction. Une valeur égale à 0 signifie que l'opération s'est bien passée. Sinon, il y a eu une erreur (4) ;

- l'ouverture ou la fermeture de consultation de l'information est accordée si l'opération s'est bien déroulée.

e) Postcondition :

- Si le code-retour est égal à 0, l'information entre ou sort du mode CONSULTATION. Sinon, l'E.R. sera dans l'état précédant l'appel à la fonction.

5. MISE-A-JOUR (utilisateur, date jour, objet, mode, code-retour)

a) Données :

- le code, la permission et le groupe de l'utilisateur ;

- la date du jour ;

- le mode d'ouverture (OUVERTURE ou FERMETURE) ;

- le type et le code de l'objet à consulter.

b) Préconditions :

- la base de données de l'E.R. doit être ouverte ;

- les données fournies en entrée doivent être valides.

c) Objectif de la fonction :

La fonction mise-à-jour a pour objectif de permettre la mise à jour, à la demande d'un utilisateur, d'une information ou d'un groupe d'informations.

Conditions à remplir :

- L'utilisateur doit avoir accès à l'objet en mise-à-jour ;

- Si l'information est en mode CONFIDENTIEL, seul le propriétaire de celle-ci peut la mettre à jour.

d) Résultats :

- code-retour : un code d'erreur est renvoyé au retour de chaque appel de la fonction. Une valeur égale à 0 signifie que l'opération s'est bien passée. Sinon, il y a eu une erreur (4) ;

- l'ouverture ou la fermeture de mise-à-jour de l'information est accordée si l'opération s'est bien déroulée.

e) Postcondition :

- Si le code-retour est égal à 0, l'information entre ou sort du mode MISE-A-JOUR. Sinon, l'E.R. sera dans l'état précédant l'appel à la fonction.

6. RECHERCHE (utilisateur, date jour, cond, res, code-retour)

a) Données :

- le code, la permission et le groupe de l'utilisateur ;
- la date du jour ;
- le type et les critères portant sur les objets à rechercher ;

b) Préconditions :

- la base de données de l'E.R. doit être ouverte ;
- les données fournies en entrée doivent être valides.

c) Objectif de la fonction :

La fonction recherche a pour objectif de permettre la recherche, à la demande d'un utilisateur, d'objets de même type ayant certaines propriétés.

Condition à remplir :

- aucune condition particulière ne doit être remplie.

d) Résultats :

- code-retour : un code d'erreur est renvoyé au retour de chaque appel de la fonction. Une valeur égale à 0 signifie que l'opération s'est bien passée. Sinon, il y a eu une erreur (4) ;
- la liste des codes des objets correspondant aux critères de la recherche.

e) Postcondition :

- Si le code-retour est égal à 0, la recherche s'est bien passée.

7. PROPRIETE (utilisateur, date jour, objet, code-retour)

a) Données :

- le code, la permission et le groupe de l'utilisateur ;
- la date du jour ;
- le type et le code de l'objet dont on veut les propriétés (6) .

b) Préconditions :

- la base de données de l'E.R. doit être ouverte ;
- les données fournies en entrée doivent être valides.

c) Objectif de la fonction :

La fonction propriété a pour objectif de donner, à la demande d'un utilisateur, l'ensemble des propriétés de l'objet dont on donne le code.

Condition à remplir :

- aucune condition particulière ne doit être remplie.

d) Résultats :

- code-retour : un code d'erreur est renvoyé au retour de chaque appel de la fonction. Une valeur égale à 0 signifie que l'opération s'est bien passée. Sinon, il y a eu une erreur (4) ;
- l'ensemble des propriétés de l'objet si celui-ci existe.

e) Postcondition :

- Si le code-retour est égal à 0, l'ensemble des propriétés est retourné dans objet.

---

(6) Le terme propriété est à prendre au sens de valeur d'attribut.

CHAPITRE 4 : IMPLEMENTATION

1. INTRODUCTION

Ce chapitre a pour objectif de donner une description de l'implémentation de l'E.R.

Tout d'abord, on verra les deux configurations matérielles sur lesquelles nous avons travaillé.

Ensuite, nous analyserons les transformations de schémas partant du schéma E-A du troisième chapitre pour arriver à un schéma dépendant du SGBD cible choisi : Ingres.

Enfin, nous terminerons par l'architecture de l'application en distinguant une architecture logique visant à travailler le plus indépendamment possible des outils d'implémentation et une architecture physique correspondant à l'adaptation de la première aux contraintes rencontrées.

2. DESCRIPTION DE LA CONFIGURATION MATERIELLE ET LOGICIELLE

Première configuration :

- Station de travail SUN-2 ;
- Système d'exploitation : UNIX ;
- Système de Gestion de Base de Données Relationnel (SGBDR) : Sun MicroIngres ;
- Langage de programmation : Equel/C.

Deuxième configuration :

- Mini-ordinateur VAX 750 ;
- Système d'exploitation : ULTRIX ;
- SGBDR : Ingres ;
- Langage de programmation : Equel/C ;
- Debugger : DBX.

Remarque :

Les deux configurations sont compatibles au niveau de leur système d'exploitation , SGBDR et langage de programmation. Nous n'avons pas utilisé d'autres outils, pour la gestion de l'interface homme-machine par exemple (Suntools), qui eux ne sont plus compatibles. La station de travail SUN utilise en effet un écran graphique bit-map avec possibilité de multi-fenêtre qu'un terminal VT220 ou VT240 ne possède pas.

3. REPRESENTATION DES DONNEES

L'E.R. étant implémenté sur un SGBDR, une transformation de schémas a du être réalisée du schéma E-A de la figure 3-1 vers un schéma conforme relationnel.

Nous nous sommes pour cela basés sur la méthode utilisée dans (HAIN84C). La figure 4-1 illustre de manière syntétique les différentes phases de transformations à opérer.

Les trois phases de transformation sont : le rappel de l'analyse fonctionnelle, la conception logique et la conception physique.

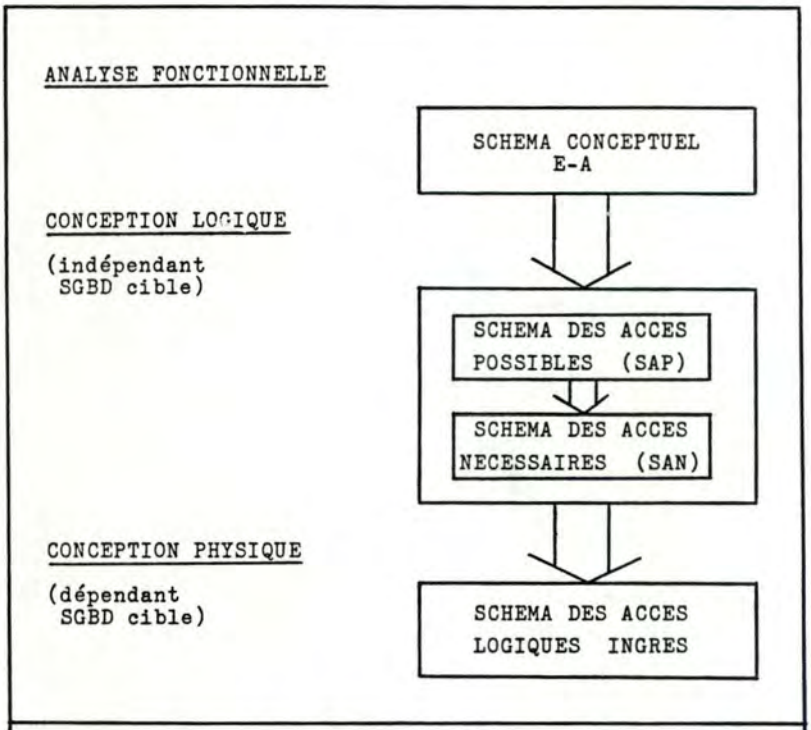


Fig. 4-1 : Transformation de schémas

3.1. Analyse fonctionnelle

Lors de l'analyse fonctionnelle, dans le troisième chapitre, nous avons choisi le modèle E-A comme modèle de structuration des données. Ce modèle servira dans ce présent chapitre comme point de départ à une série de transformations pour la conception de la base de données.

A titre de rappel, la figure 4-1 illustre le schéma E-A de l'E.R.

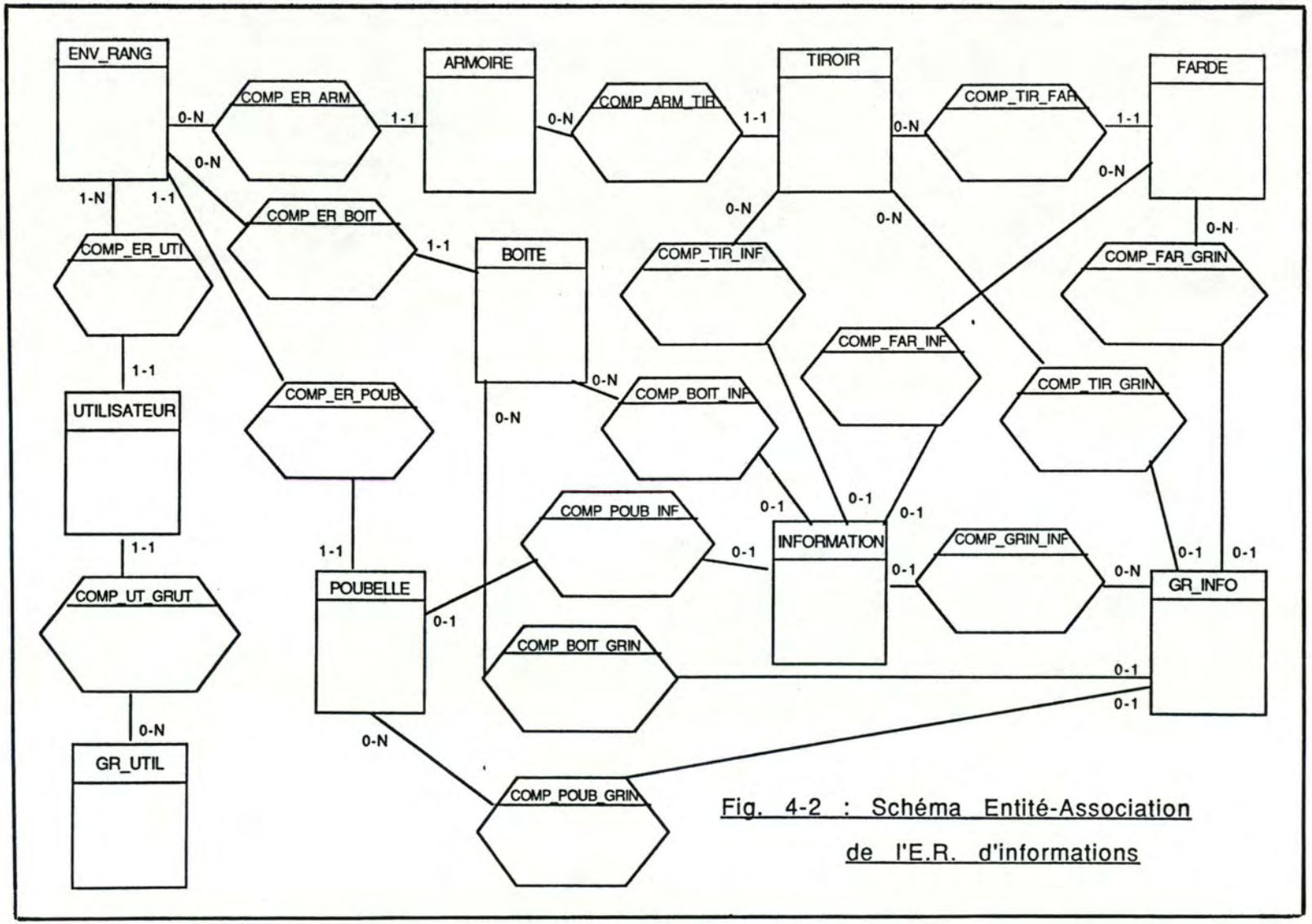


Fig. 4-2 : Schéma Entité-Association de l'E.R. d'informations

3.2. Conception logique

La phase de conception logique de l'analyse des données utilise le Modèle d'Accès Généralisé (HAIN81) comme intermédiaire entre le modèle conceptuel et le SGBD proprement dit.

Une particularité importante de cette phase intermédiaire est l'indépendance de tout SGBD.

Cette phase est décomposée en deux sous-phases : une première qui aboutit au schéma des accès possibles (SAP) et une seconde qui aboutit au schéma des accès nécessaires (SAN).

A. Schéma des accès possibles

Ce premier schéma exprime de la manière la plus complète possible le schéma conceptuel en n'accordant pas encore d'importance aux accès qui seront demandés aux objets de l'E.R.

La figure 4-3 illustre le schéma des accès possibles.

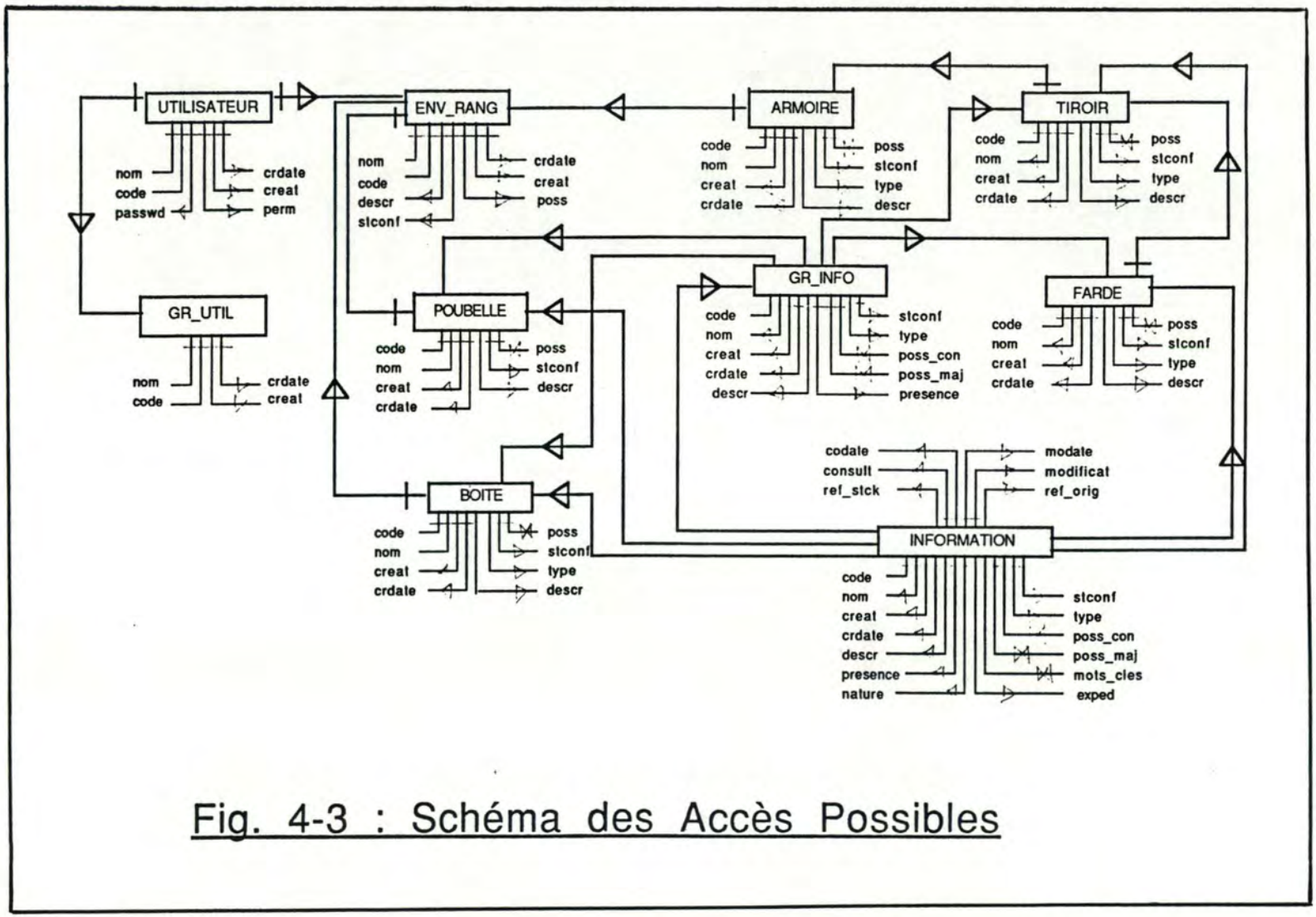


Fig. 4-3 : Schéma des Accès Possibles

B. Schéma des accès nécessaires

Ce schéma correspond au schéma des accès possibles dans lequel on fait ressortir les accès qui seront demandés par les opérations sur les objets.

Pour aboutir à ce schéma, une étude des algorithmes qui accèdent aux données doit normalement être effectuée. Une analyse détaillée de ces algorithmes ne sera pas réalisée dans le cadre de ce mémoire. Pour généraliser les accès nécessaires aux objets de l'E.R., trois types d'accès seront retenus :

- accès à un objet par son code ;
- accès au contenu d'un objet ;
- accès au contenant d'un objet.

La figure 4-4 illustre le schéma des accès nécessaires.

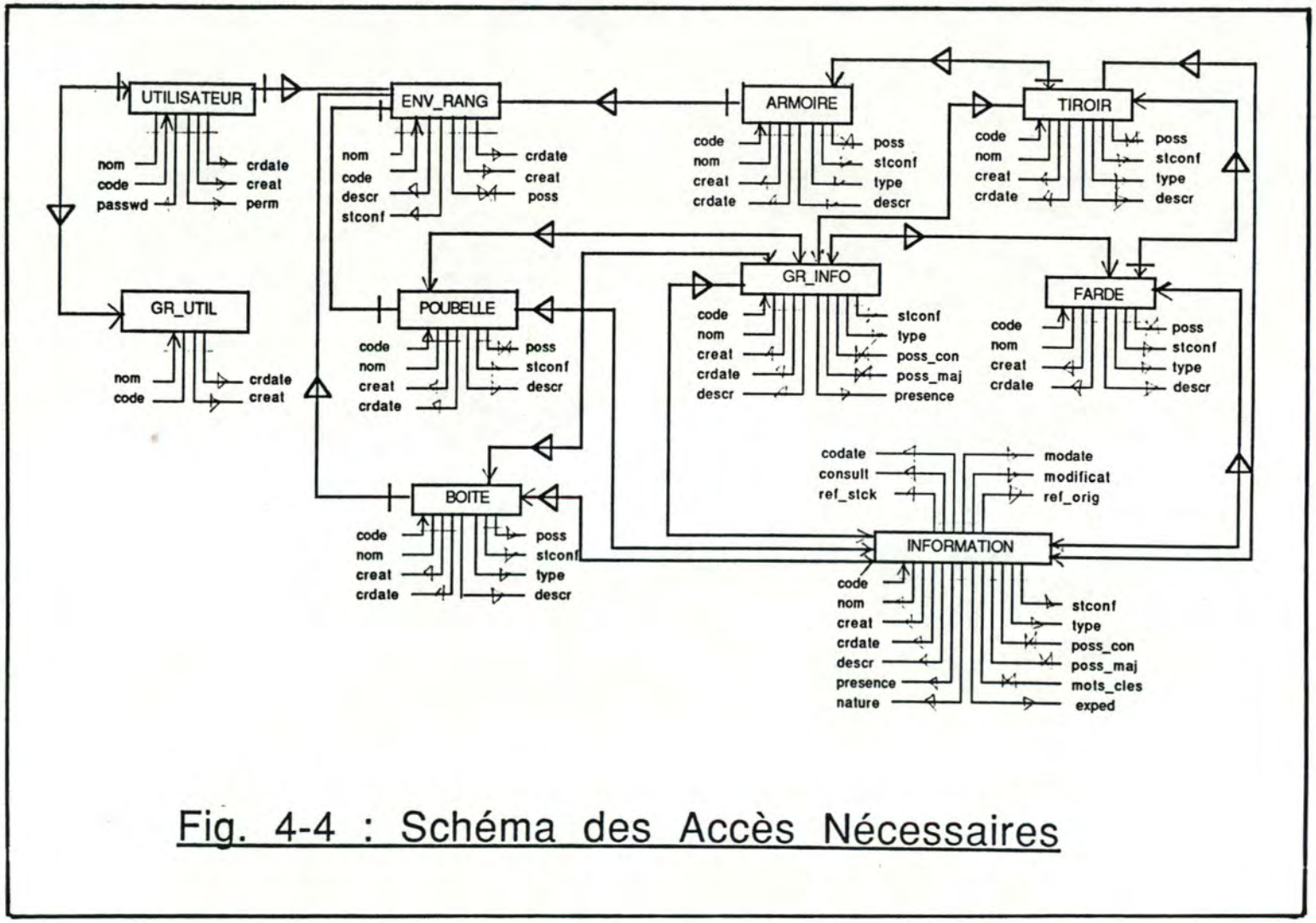


Fig. 4-4 : Schéma des Accès Nécessaires

3.3. Conception physique

La phase de conception physique consiste à traduire le schéma des accès nécessaires en un schéma qui soit conforme au SGBD cible choisi. Dans ce cas il s'agit du SGBDR Ingres

La figure 4-5 illustre le schéma des accès logiques Ingres.

Quelques remarques à propos de ce schéma :

1. Il n'a pas été nécessaire de transformer les items répétitifs du schéma des accès nécessaires étant donné que ces items ont été implémentés sous forme de chaînes de caractères et que Ingres offre des facilités pour le traitement de ce type de donnée. Il s'agit des items poss, poss-con, poss-maj et mots-cles.

2. Certaines clés d'accès n'ont pas été retenus pour les tables qui ne contiennent qu'une seule ligne. Le temps d'accès sur ces type d'article n'étant pas dégradé par un accès séquentiel. Il s'agit des items code des types d'article : ENV-RANG et POUBELLE.

3. Les chemins d'accès ayant comme origine ou extrémité le type d'article ENV-RANG n'ont pas été retenus pour la transformation étant donnés que dans la version actuelle, il n'existe d'un seul E.R.

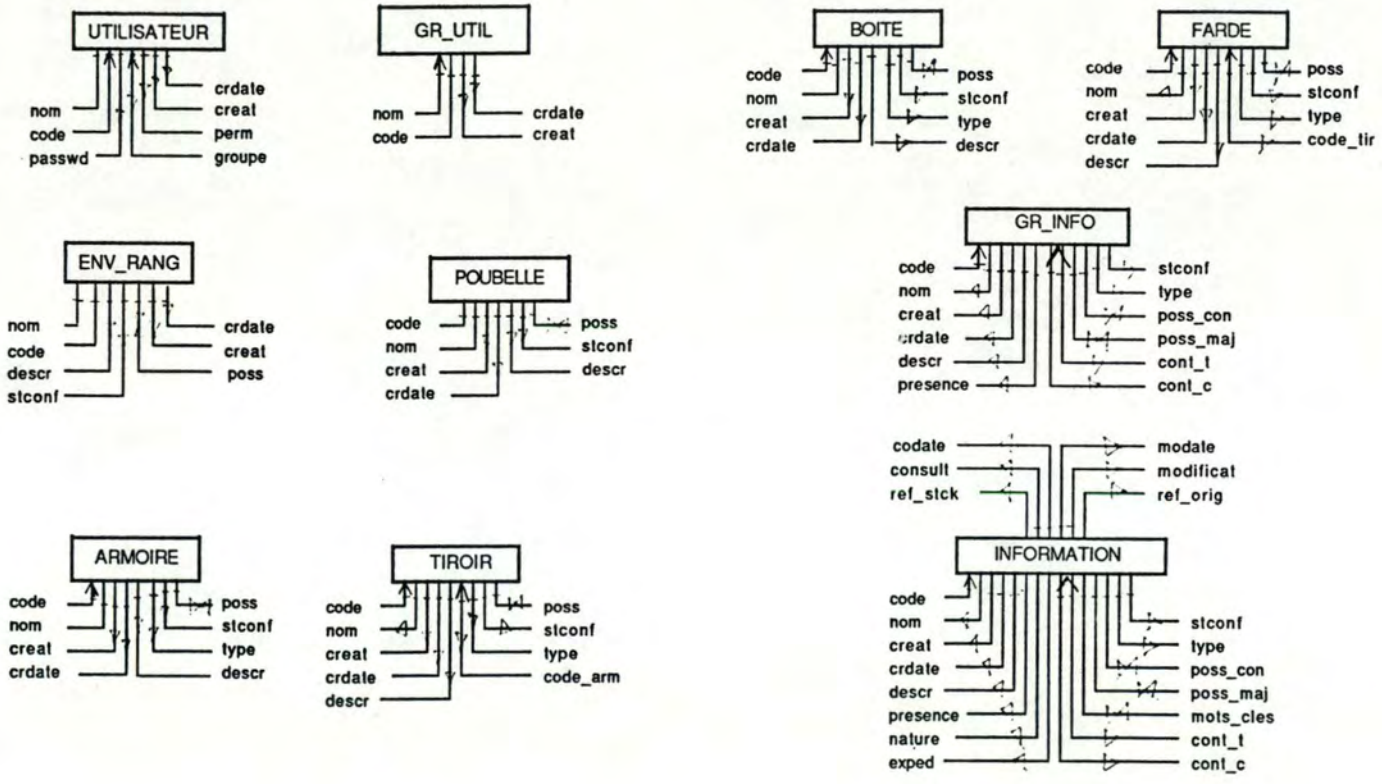


Fig. 4-5 : Schéma des Accès Logiques Ingres

4. ARCHITECTURE DE L'APPLICATION

On distinguera deux architectures : l'architecture logique et l'architecture logique.

4.1. ARCHITECTURE LOGIQUE

L'architecture logique correspond à la représentation des différentes couches d'abstraction de l'application d'après la méthode utilisée dans (VANL85).

Rappelons que l'implémentation de l'E.R., dans le cadre de ce mémoire, se limite à la gestion des objets de celui-ci. Nous avons procédé à une découpe en deux couches de manière à isoler les fonctions dépendant du SGBD Ingres. La figure 4-6 illustre cette découpe.

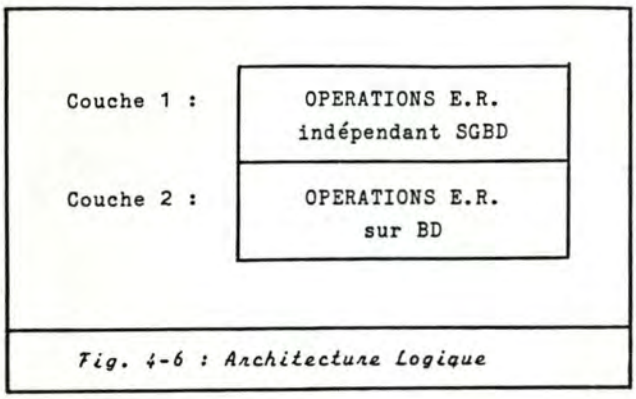


Fig. 4-6 : Architecture Logique

La couche supérieure représente les différentes opérations, qui ont été décrites dans le troisième chapitre lors de l'analyse fonctionnelle, au niveau des différentes vérifications des conditions à repecter.

La couche inférieure représente l'exécution de l'opération proprement dite sur le SGBD Ingres. Cette couche cache donc les accès au SGBD. Ainsi, lors d'un changement éventuel de SGBD, seul la couche inférieure est à respécifier.

4.2. ARCHITECTURE PHYSIQUE

Comme l'illustre la figure 4-7, l'architecture physique de l'application reflète l'architecture logique pour ses deux couches supérieures.

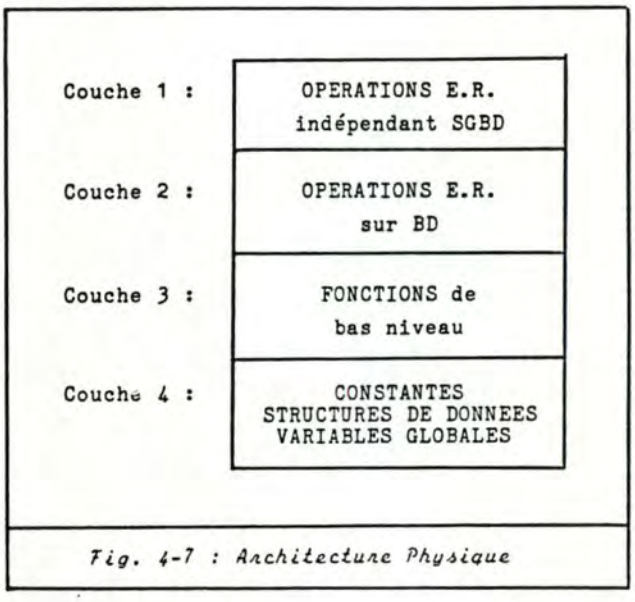


Fig. 4-7 : Architecture Physique

Néanmoins, deux couches supplémentaires viennent s'y ajouter :

- une première couche renferme les fonctions de bas niveau appelées à la fois par la couche 1 et la couche 2.

- une deuxième couche renferme les déclarations des constantes, des structures de données et des variables globales utilisées par les trois couches supérieures.

La première couche correspond aux fichiers : CREATION.C, SUPPRESSION.C, MODIFICATION.C, CONSULTATION.C, MISE_A_JOUR.C, RECHERCHE.C et PROPRIETE.C de l'annexe A.

La deuxième couche correspond aux fichiers : CREAT.QC, SUPPR.QC, MODIF.QC, CONSUL.QC, MAJ.QC, CONTEN.QC et RECH.QC de l'annexe A.

La troisième couche correspond aux fichiers : LIB_F.QC et LIB_I.AC de l'annexe A.

La quatrième couche correspond aux fichiers : LIB_V.QC et LIB_C.QC de l'annexe A.

5. CONCLUSION

Ce chapitre a permis de tracer une description de l'implémentation de l'E.R.

Au niveau des données, il y a eu des transformations du schéma E-A vers un schéma conforme au SGBD Ingres.

Au niveau des opérations, il y a eu la description de l'architecture logique et physique de l'application.

CHAPITRE 5 : EVALUATION.

1. INTRODUCTION.

Dans ce chapitre, nous essayerons de présenter une évaluation la plus complète possible du travail effectué au travers des différentes phases de la démarche que nous avons suivie.

Cette évaluation est le résultat de l'expérience acquise par l'analyse et la réalisation d'un environnement automatisé de rangement d'informations.

Certains domaines informatiques nouveaux seront abordés et présentés comme des alternatives possibles aux techniques que nous avons utilisées. Ils ne seront examinés que partiellement, dans la mesure où ils pallient aux inconvénients rencontrés dans les choix que nous avons effectués. L'objectif de ce chapitre sera donc de prendre une position critique vis-à-vis du travail réalisé tout en introduisant de nouvelles idées qui pourront éventuellement servir de prolongements au mémoire.

2. MODELE D'E.R. D'INFORMATIONS.

A. Avantage.

Nous avons pris comme modèle de départ d'E.R., celui d'un E.R. manuel : le bureau. Ce choix a été fait pour répondre à un esprit bureautique car nous pensons qu'une des raisons qui a rendu la percée de la bureautique difficile en ses débuts, était un manque d'adaptation à l'utilisateur "non informaticien" tant au niveau matériel que logiciel.

Comme nous l'avons déjà montré dans l'introduction du mémoire, certains progrès apparaissent par le concept de station de travail.

Un effort est également nécessaire en ce qui concerne les applications . Le bureau en tant que modèle de rangement représente pour nous une prise en considération de l'utilisateur dans la mesure où la plupart des concepts employés sont déjà connus et manipulés.

Du point de vue de l'informaticien, ce problème de l'adaptation peut être négligeable puisque celui-ci est en contact permanent avec le matériel informatique. Par contre, l'utilisateur occasionnel qui ne comprend pas toujours ce qu'il y a "derrière" l'ordinateur, a du mal à s'imaginer ce que représente un "fichier" informatique. On lui dira que ces fichiers peuvent contenir des "programmes" ainsi que des "données" sans qu'il y ait apparemment de différence extérieurement. De plus si on lui demande d'effectuer des "back-up" parce que les données peuvent "se perdre",

l'informatique devient pour lui un obscur et mystérieux objet. Alors qu'il suffirait de parler le même langage que lui : armoire ou bac à fiches au lieu de fichier, et d'élaborer une gestion automatique et transparente de sauvegarde des fichiers comme le permettent certains traitements de texte.

B. Inconvénient.

Cette comparaison avec la manière de travailler manuelle n'est pas toujours bénéfique ou peut même être déconseillée dans certains cas.

Ainsi, nous avons écarté dès le départ le concept d'étagère parce que celle-ci ne se distinguait de l'armoire que par son apparence extérieure. En effet, cette distinction n'existe plus dans un E.R. automatisé.

L'expérience serait susceptible de montrer qu'il pourrait en être de même pour d'autres concepts. Ainsi, pour le concept de boîte, nous avions pris sa taille comme distinction majeure, par rapport à l'armoire, c'est-à-dire le nombre d'informations qu'elle pouvait contenir. Dans le domaine informatique ce critère n'a plus nécessairement la même importance, les fichiers n'étant pas limités en taille, ou du moins ne le sont qu'à concurrence de la capacité du support sur lequel ils sont enregistrés.

C. Généralisation.

Après l'expérience acquise par l'étude que nous avons faite des types d'objet constituant un E.R. et si nous abandonnons l'optique bureautique qui a consisté à prendre comme modèle de départ le bureau, nous pourrions aboutir à un schéma entité-association (E-A) différent de celui que nous avons utilisé au chapitre 3 (figure 3-1).

En effet, si l'on ne considère pas les concepts d'utilisateur et de groupe d'utilisateurs, on pourrait généraliser ce schéma comme l'illustre la figure 5-1.

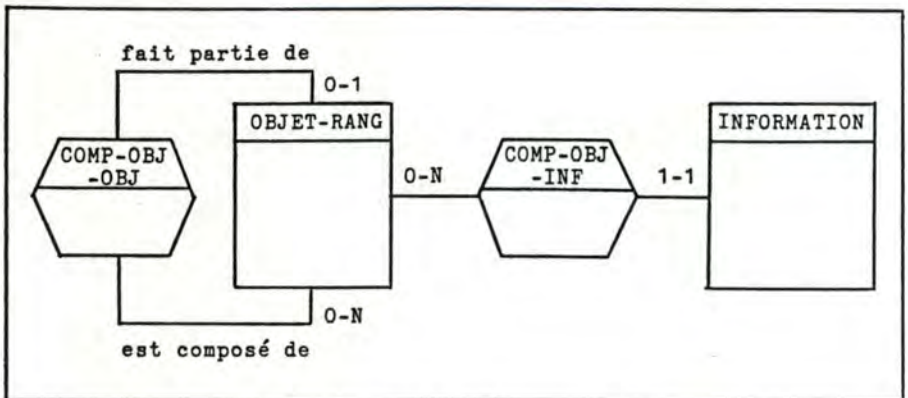


Fig. 5-1 : Généralisation de l'E.R.

Notons que pour que ce schéma puisse être sémantiquement équivalent à celui que nous avions utilisé, certains attributs doivent être ajoutés. Par exemple, OBJET_RANG devrait être complété par un attribut représentant le type de l'objet : armoire, tiroir, boîte, ... Un nombre important de contraintes d'intégrité supplémentaires devraient de plus être gérées notamment sur les valeurs des attributs de OBJET_RANG. Nous ne les développerons pas ici.

Nous voyons que l'E.R. peut être représenté de deux façons au moyen du modèle E-A. Il est intéressant de se poser la question suivante : l'utilisation du modèle E-A est-elle réellement indépendante du type de public auquel ce modèle est destiné ?

Dans le cadre du cours d'analyse fonctionnelle (BOPI84) donné en première licence, on nous a présenté le modèle E-A comme un outil d'analyse d'organisation, l'objectif étant de pouvoir montrer aux dirigeants et employés d'une entreprise la structure souvent complexe des informations qu'ils manipulent. Cet outil était donc destiné à être compris par des personnes non initiées à l'informatique.

Dans le cadre du cours de conception de fichiers et banques de données (HAIN84C), on nous a présenté le même modèle E-A comme modèle initial à la conception de bases de données. Ce modèle E-A était donc vu comme un modèle de représentation de données destiné à être utilisé par des concepteurs de banques de données supposés posséder une formation en informatique.

En réponse à la question ci-dessus, le schéma de la figure 3-1 semble idéalement destiné à un public "non informaticien". Le concepteur de bases de données, par contre, portera peut-être une préférence pour le schéma de la figure 5-1 qui paraîtra certainement trop abstrait pour la plupart des non initiés à l'informatique.

3. MODELE CONCEPTUEL DE REPRESENTATION DES DONNEES.

A. Avantage.

Le modèle conceptuel E-A, que nous avons choisi pour la représentatation des informations, présente comme principal avantage, en dehors du fait qu'il soit indépendant de tout SGBD cible, de permettre des transformations relativement aisées vers des schémas conformes aux SGBD cibles choisis.

Ces transformations se font en plusieurs phases comme on l'a vu au quatrième chapitre : Implémentation. Il faut faire appel à des règles de transformation et respecter des contraintes en fonction de la famille du SGBD et du SGBD cible choisi.

Une analyse et un développement d'un support méthodologique et logiciel assistant la production de solution COBOL, CODASYL et SQL au départ d'un schéma E-A passant par une solution logique exprimée dans le MAG, a notamment fait l'objet d'un mémoire (CHMU86).

B. Inconvénients.

Par rapport au modèle E-A de base décrit dans (BOPI83), la présente application a fait ressortir quelques inconvénients de ce modèle.

1. Pour éviter certains problèmes de représentation graphiques du schéma E-A, nous avons représenté certaines associations par des attributs accompagnés de contraintes d'intégrité (cfr. chapitre 3. 2.2.4.2.).

2. Au niveau de l'analyse des attributs des T.E., nous avons dû regrouper les T.E. pour éviter une répétition dans les interprétations. Rappelons qu'à cet effet, les types d'objet avaient été regroupés en trois groupes : les types d'objet physiques, les types d'objets logiques et les utilisateurs. Chacun de ces groupes étant caractérisés par une ressemblence dans leurs propriétés.

Ces deux points révèlent donc un problème de lourdeur dans la représentation et de répétition dans l'analyse des informations.

Pour montrer comment le problème de répétition peut être géré au niveau conceptuel, dans le point suivant est introduit un modèle alternatif de représentation de données.

C. Alternative : les réseaux sémantiques.

Sans entrer dans une étude appronfondie d'autres modèles de représentation de données, voyons comment on pourrait résoudre le problème de la répétition des attributs à l'aide des réseaux sémantiques.

Reprenons tout d'abord quelques définitions se rapportant aux réseaux sémantiques.

"L'objet des réseaux sémantiques est de décrire les éléments d'un univers et leurs interrelations".

"Les deux composants du modèle sont les noeuds signifiant la présence d'objets, de concepts, de sit_uations ou d'événements, et les arcs (orientés) exprimant leurs interrelations." (LEON86)

En prenant le cas de la représentation des types d'objet : armoire, boîte et poubelle, on obtient le réseau sémantique correspondant à la figure 5-2.

Cet exemple simplifié de réseau fait apparaître trois types de particularités :

- *"il autorise des inférences : la relation est-un signifie qu'un concept forme un sous-ensemble d'un autre". Par exemple : ARMOIRE est-un OBJET-DE-RANGEMENT. Ceci implique que toutes les propriétés de OBJET-DE-RANGEMENT sont également propriétés d'ARMOIRE ; il y a ce qu'on appelle héritage des propriétés ;*

- *il permet la représentation d'occurences d'objets au même titre que les types d'objets. Ainsi armoire-X est une occurrence du type d'objet ARMOIRE par la relation est-élément-de ;*

- *il ne demande qu'une seule déclaration des concepts. Par exemple : le concept POSSIBILITE-D'ACCES sera déclaré une fois et tout autre concept voulant l'utiliser pourra y référer en pointant un arc orienté muni du nom d'une relation.*

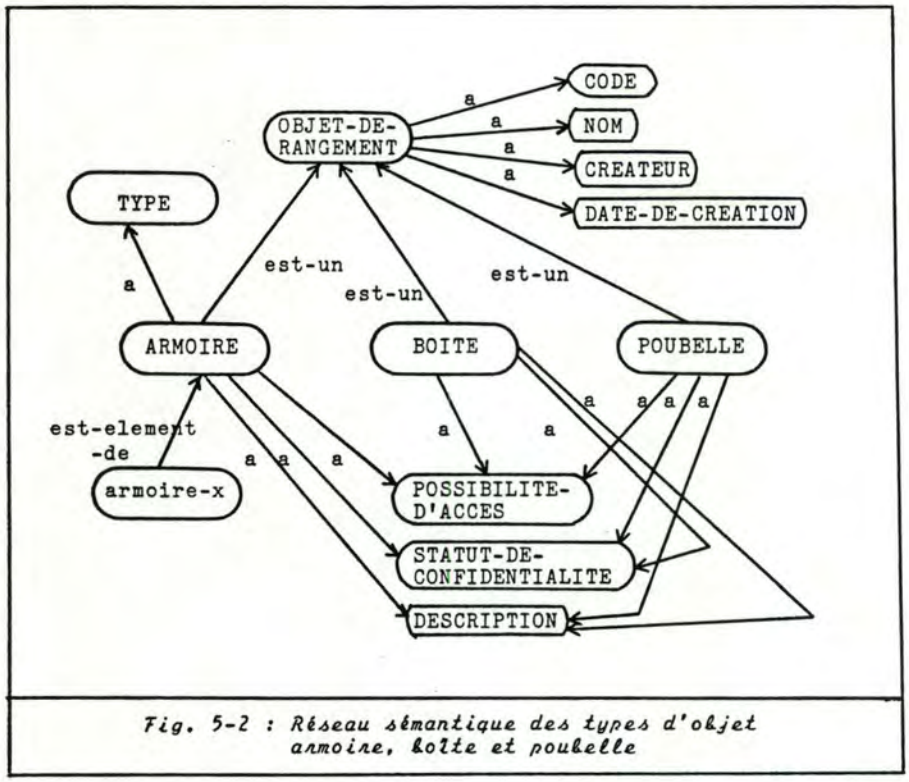


Fig. 5-2 : Réseau sémantique des types d'objet armoire, boite et poubelle

Nous n'évaluerons pas plus en profondeur ce modèle dans le cadre de ce mémoire. Signalons simplement que ce modèle possède également certains inconvénients. Une analyse de ces inconvénients a d'ailleurs été faite dans (LEON86).

4. OUTILS D'IMPLEMENTATION.

Les outils d'implémentation que nous avons utilisé , en dehors de la machine et du système d'exploitation, sont : l'environnement de programmation C et du SGBDR Ingres.

A. Avantages du langage C.

Nous ne passerons pas en revue tous les avantages du langage C, la liste en serait trop longue et ce n'est pas l'objet de cette évaluation. Signalons simplement qu'il nous a permis de travailler de manière modulaire en donnant la possibilité de cacher un maximum d'information à l'intérieur des fonctions. Seuls le nom de la fonction, les paramètres et le respect des pré-conditions sont nécessaires pour utiliser celles-ci.

B. Inconvénients du langage C.

Un premier inconvénient mineur est l'adaptation au passage des paramètres par adresse. Quand on a l'habitude de travailler en Pascal où ce mécanisme existe mais de manière transparente, on rencontre quelques problèmes. Ceci semble d'ailleurs être le cas pour un bon nombre de personnes qui s'initient au langage C.

Un autre inconvénient, qui n'est pas vraiment particulier au langage C mais qui se retrouve dans tous les langages typés comme le Pascal, certains Basic évolués ou le C, se pose lors du passage des paramètres. Le fait qu'un paramètre doive être d'un type spécifique ne permet pas de passer n'importe quelle donnée comme nous aurions aimé le faire pour nos opérations. En effet, pour minimiser le nombre d'opérations sur l' E.R., nous avons pris l'option de n'avoir qu'une opération distincte pour l'ensemble des types d'objet, telle que par exemple :

CREATION (utilisateur, objet, code-erreur) ;

Le fait que le paramètre objet doive avoir un type bien défini nous a posé un problème au départ. Il existe une solution pour le langage C qui ne peut malheureusement pas être transposée au langage Pascal (figure 5-3).

Pour accéder à un objet, il est tout d'abord nécessaire de vérifier son type (objet.t) qui permettra de déduire les propriétés de l'objet. Par exemple, si l'objet est une information, on sait qu'elle aura une propriété expédition accessible par objet.o.i.expédition.

Cette solution permet donc de passer une variable par paramètre qui contient n'importe quel objet de l'E.R.

```
union
{     struct ENV-RANG e ;
      struct ARMOIRE  a ;
      struct TIROIR   t ;
      struct INFORMATION i ;
                :
} ;

typedef STRUCT-OBJ
{     char t ;
      union type-objet o ;
}     objet ;
```

Fig. 5-3 : Solution en C : union

Un mécanisme similaire existe en Cobol où l'on redéfini plusieurs fois la même zone mémoire à des variables différentes. Là aussi, il doit y avoir une zone fixe (TYPE-OBJET) et une zone variable (CORPS-OBJET) qui peut contenir n'importe quel objet.

La figure 5-3b illustre la solution que le langage Cobol propose.

```
01 OBJET
   02 TYPE-OBJET PIC X
   02 CORPS-OBJET PIC X (500)
   02 INFORMATION REDEFINES CORPS-OBJET
      03 CODE  9(6)
      03 NOM   X(25)
      03 CREATEUR 9(6)
           :
   02 ARMOIRE REDEFINES CORPS-OBJET
           :
   02 UTILISATEUR REDEFINES CORPS-OBJET
           :
```

Fig. 5-3b : Solution COBOL : Redéfinition structurée d'une zone de variable

Rappelons que dans les deux cas, le mécanisme de la zone variable est à gérer par le programmeur.

C. Avantage d'Ingres.

L'avantage principal que nous retiendrons d'Ingres est qu'il a permis d'implémenter les opérations sur l'E.R. en un

minimum d'instructions. Il s'agit réellement d'un langage de haut niveau pour la gestion de données.

D. Inconvénients d'Ingres.

1. Il nous a semblé qu'Equel/c, le langage d'Ingres, ne permettait pas d'utiliser toutes les facilités normalement accessibles au langage C.

Ainsi, les variables interface entre le programme C et les requêtes Ingres ne peuvent faire plus d'un niveau de structuration et sont automatiquement considérées comme globales par le pré-processeur d'Ingres. Ceci oblige la déclaration de certaines variables globales au programme. Le problème des niveaux de structuration, au nombre de trois pour les objets (exemple : objet.o.i.expédition), nous a obligé à faire une conversion vers une variable à un seul niveau avant de pouvoir l'utiliser à l'intérieur d'une requête Ingres. Cette conversion a cependant été cachée dans la couche qui s'occupe des opérations dépendantes du SGBDR Ingres.

2. Signalons qu'en Ingres, il n'est pas possible d'imbriquer les requêtes. Comme le montre la figure 5-4 (a), si l'on veut que la requête-2 utilise le résultat de la requête-1, il faut décomposer les 2 requêtes vers la forme (b). Le résultat de la requête-1 sera passé vers la requête-2 par une gestion de liste ou de tableau.

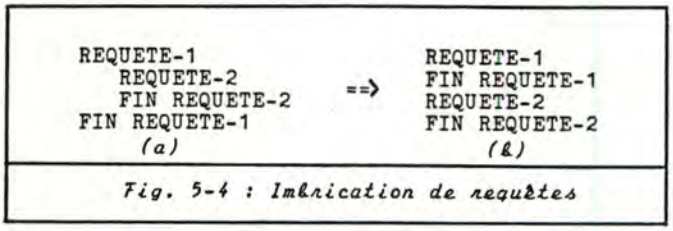


Fig. 5-4 : Imbrication de requêtes

E. Alternative : Lisp

Nous allons présenter un langage alternatif à Equel/C qui présente certaines caractéristiques particulières sur lesquelles il nous paraît intéressant de s'arrêter. Il s'agit de Lisp.

Premièrement, le langage fonctionnel Lisp possède la particularité de travailler avec des données à la fois simples et générales. Une donnée est soit un atome, soit une liste. A partir de ces deux concepts, on peut construire toutes les structures de données désirées sans avoir les inconvénients des langages typés cités au point B pour le passage des paramètres d'une fonction.

Deuxièmement, la langage Lisp ne distingue pas syntaxiquement les programmes de données. Une fonction en Lisp

a la forme d'une liste dont le premier atome est le nom de la fonction et les atomes ou listes suivants sont les paramètres de celle-ci.

Troisièmement, à partir de la deuxième particularité, un programme en Lisp peut se modifier dynamiquement c'est-à-dire qu'il peut modifier ou générer une fonction pour l'utiliser dans la suite de son exécution. Il s'agit du concept de macro qui correspond à quatre phases habituellement distinctes en programmation classique (figure 5-5).

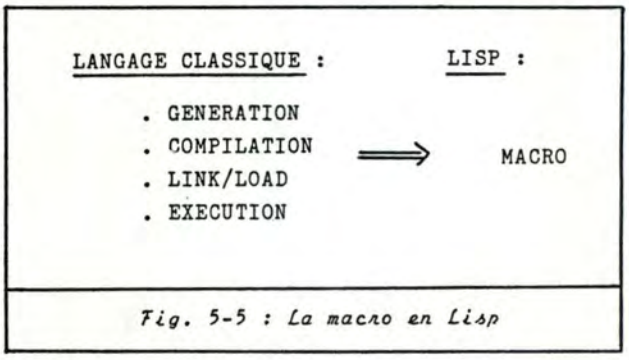


Fig. 5-5 : La macro en Lisp

L'exemple suivant présente la génération d'une requête 'RETRIEVE' en Ingres à l'aide de la fonction genere écrite en Lisp. La figure 5-6 illustre la déclaration de la fonction genere et la figure 5-7 en donne un exemple d'utilisation.

```
(DEFUN GENERE
  (LAMBDA (OBJ PAR)
    (FORMATE
      (LIST
        (SUBST 'RES (CADR OBJ)
          (SUBST 'TOBJ (CAR OBJ)
            '(RETRIEVE RES TOBJ CODE) ))
        (SUBST 'TOBJ (CAR OBJ)
          (SUBST 'PROP (CAAR PAR)
            (SUBST 'OP1 (CADAR PAR)
              (SUBST 'VAL (CAR (CDDAR PAR))
                '(WHERE TOBJ PROP OP1 VAL) ))))
        (MAPCAR
          '(LAMBDA (ELEM)
            (SUBST 'OP2 (CAR ELEM)
              (SUBST 'TOBJ (CAR OBJ)
                (SUBST 'PROP (CADR ELEM)
                  (SUBST 'OP1 (CADDR ELEM)
                    (SUBST 'VAL (CAR (CDDDR ELEM))
                      '(OP2 TOBJ PROP OP1 VAL) ))))))
          (CADR PAR) )))))
```

Fig. 5-6 : Fonction genere

COMMANDES EXECUTEES :

```
(SETQ OBJET '(INFORMATION RESULTAT))
(SETQ CRITERES '((TYPE = "FACTURE")
                 (OR CRDATE >= "01-01-86)
                 (OR CREATEUR = 693) )))
(GENERE OBJET CRITERES)
```

TEXTE GENERE :

```
RETRIEVE (RESULTAT=INFORMATION.CODE)
WHERE INFORMATION.TYPE="FACTURE"
      OR INFORMATION.CRDATE>= "01-01-86"
      OR INFORMATION.CREATEUR= 693
```

Fig. 5-7 : Exemple d'exécution de genere

La fonction genere utilise deux paramètres d'entrée. Le premier porte sur le type de l'objet à retrouver ainsi que la variable qui contiendra le résultat, et le second sur les conditions à remplir par l'objet candidat à la recherche.

La philosophie de la fonction genere se base sur la forme générique de la requête à générer. Cette forme est décomposée en 3 parties :

1. (RETRIEVE RES TOBJ CODE) toujours présent ;

2. (WHERE TOBJ PROP OP1 VAL) toujours présent ;

3 (OP2 TOBJ PROP OP1 VAL) peut être représenté 0,1 ou plusieurs fois suivant le nombre de conditions de la requête. La fonction mapcar permet de gérer cette multiplicité.

La fonction subst permet de substituer un atome par un autre dans une liste.

La fonction genere consiste à remplacer les mots de cette requête générique par ceux représentant la requête à générer et à formater le tout sous la forme d'une requête Ingres réalisé par la fonction formate.

Les sources complètes des différentes fonctions utilisées dans genere sont reprises à l'annexe C.

Cet exemple ne permet cependant pas encore de montrer la troisième particularité de Lisp. On est jusqu'à présent seulement parvenu à générer une requête Ingres, pas encore à l'exécuter. Ceci n'a pas été possible car dans notre version de Lisp, il n'y avait pas d'interface entre le langage Lisp et du SGBDR Ingres. A notre connaissance, il n'existe d'ailleurs pas de version Lisp présentant cette interface. Il s'agit en fait d'un problème de compilation qui n'est pas particulier au langage. De même que l'interface Ingres existe pour des langages classiques comme Cobol, Fortran ou Pascal, elle serait tout aussi envisageable pour d'autres langages. A ce titre nous avons eu connaissance qu'une interface Ingres avait été réalisée pour une version de Prolog sur une station de travail SUN (BIM85). Pourquoi ne serait-ce donc pas possible aussi pour Lisp ?

Cet exemple montre donc qu'il serait possible de générer automatiquement et dynamiquement des requêtes Ingres sur mesure, ce qui n'est pas toujours le cas en Equel/C. De plus cette génération automatique permettrait de travailler sur des objets ou des propriétés d'objet non prévu initialement sans devoir modifier les programmes.

5. PERFORMANCES DE L'IMPLEMENTATION.

A. Réalisation.

Nous n'avons pas, dans le cadre de ce mémoire, insisté sur la performance de l'implémentation. L'objectif était en effet de d'abord montrer la faisabilité de l'application et la manière de la réaliser.

Nous avons cependant veillé un minimum à la performance au niveau des accès aux données. Nous avons à cet effet, comme nous l'avons déjà développé au chapitre 4, implémenté une clé calculée sur le code de chaque type d'objet. Un index secondaire sur le code du contenant des types d'objet.

Sur base de cette implémentation, la figure 5-8 reprend quelques chiffres de temps d'accès moyens observés lors des tests des différentes opérations sur l'E.R. Les programmes de tests ainsi que leurs résultats sont repris à l'annexe B.

Ouverture de la base de données er :	9-12 sec.
Fermeture de la base de données er :	2-4 sec.
Création d'un objet :	2-3 sec.
Suppression d'un objet :	3-5 sec.
Modification d'un objet :	2-3 sec.
Recherche d'un objet :	1-2 sec.

Fig. 5-8 : Temps d'accès moyens des différentes opérations sur les objets de l'E.R.

On peut remarquer que l'opération d'ouverture de la base de données (BD) est onéreuse en temps d'accès. C'est pourquoi on a posé comme pré-condition à l'appel de toute opération que la BD soit ouverte. Cela devrait éviter de devoir l'ouvrir puis la refermer à chaque appel d'une opération. Une application utilisatrice de l'E.R. n'aurait qu'à ouvrir la BD en début de session de travail et à la refermer à la sortie du programme.

Nous avons également constaté que la première opération d'une série demande en moyenne de une à deux secondes d'accès supplémentaires par rapport aux suivantes.

Signalons enfin que les tests ont été effectués sur un E.R. ne contenant qu'une quarantaine d'objets. Les chiffres avancés ne sont donc donnés qu'à titre illustratif et doivent être adaptés à la taille de l'E.R. Cette indication est particulièrement importante pour l'opération RECHERCHE.

B. Améliorations.

Pour encore améliorer la performance de l'implémentation deux types de modifications sont proposées :

1. Certains types d'objets pourraient être regroupés sous une seule table. Cette modification permettrait essentiellement de diminuer le nombre de tables représentant l' E.R. En effet, les tables ENV-RANG et POUBELLE ne comptent actuellement qu'une seule ligne chacune.

2. Les types d'objets pourraient être allégés (taille de record) en les décomposant sur plusieurs tables. On ne garderait que les champs les plus souvent utilisés en accès primaire, et d'autres champs, tel que le commentaire d'un objet, en accès secondaire c'est-à-dire dans une table séparée qui ne serait utilisée que quand on en a strictement besoin. Ceci permettrait de diminuer le temps moyen d'accès aux informations.

6. PERSPECTIVES POUR LE MEMOIRE

Il serait intéressant de voir réaliser dans l'avenir l'utilisation de l'implémentation de l'E.R. dans une application.

C'est une réalisation qui devrait d'ailleurs être concrétisée dans le cadre d'un prochain mémoire basé sur l'intégration de l'E.R. et la gestion d'une interface homme-machine de type image. On aurait la visualisation des différents objets de l'E.R. avec la possibilité d'exercer des opérations telles que : ouvrir un tiroir, visualiser son contenu, ajouter une information à un dossier, ...

D'autres idées sont ouvertes spécialement à partir de ce chapitre, que ce soit à travers d'autres langages de programmation, par exemple Smalltalk ou plutôt un enrichissement de l'E.R. qui a été réalisé dans ce mémoire.

Citons notamment les possibilités qui restent ouvertes du côté de l'analyse du contenu de l'information ou, par exemple, l'intégration d'un courrier électronique sur l'E.R.

7. CONCLUSION.

A travers ce chapitre, nous avons essayé de tracer une évaluation complète du travail. Pour cela, nous avons choisi comme philosophie d'analyser chaque phase du développement de celui-ci.

Nous avons tout d'abord justifié le modèle d'E.R. initial choisi, par les avantages que globalement il présente.

Ensuite, nous avons procédé à l'évaluation du modèle de représentation de données en fonction de inconvénients rencontrés dans l'application. Une alternative de modèle de représentation de données a été proposée sous forme d'un réseau sémantique, dans l'optique de répondre à ces inconvénients.

L'implémentation proprement dite de l'E.R. a également fait l'objet d'une analyse au niveau de l'utilisation du langage de programmation C et du SGBDR Ingres. Une introduction à quelques particularités d'un autre langage de programmation, Lisp, a été dressée en vue d'ouvrir d'autres possibilités d'implémentation de l'E.R.

En ce qui concerne la dernière phase du travail, une présentation des performances des temps d'accès a été dressée sur base de mesures effectuées lors des tests. A partir de l'interprétation de ces chiffres, quelques indications ont été données pour permettre d'encore améliorer les temps d'accès des opérations.

CONCLUSION

En guise de conclusion, rappelons l'objectif initial de ce mémoire.

A partir de la description d'un environnement de rangement manuel d'informations, le bureau, nous avons essayé de modéliser un environnement de rangement automatisé. Nous avons retenu pour cela trois groupes de types d'objet :

- un premier, appelé types d'objet physiques comprend : l'environnement de rangement, l'armoire, le tiroir, la boîte, la poubelle et la farde ;

- un deuxième, appelé types d'objet logiques comprend : l'information et le groupe d'informations ;

- un troisième est constitué de l'utilisateur et du groupe d'utilisateurs.

Nous avons ensuite réalisé l'analyse fonctionnelle de ce modèle au niveau des données et des traitements.

Nous avons aussi décrit l'implémentation que nous avons choisie au niveau de la configuration matérielle et logicielle.

Le cinquième chapitre nous semble important dans la mesure où il nous a permis de porter un jugement critique sur notre travail tant au niveau des choix que nous avons faits que des techniques mises en oeuvre. Nous avons dans la mesure du possible proposé certaines alternatives aux techniques utilisées de manière à répondre aux inconvénients que nous avons rencontrés. Nous avons enfin terminé par quelques pistes qui permettraient une continuation de ce mémoire.

BIBLIOGRAPHIE :

(ADRO83) *Smalltalk-80*
The language and its implementation
Goldberg Adele, David Robson
Addison-Wesley, 1983.

(ANFI85) *The Smalltalk programming language :*
A introduction to object-oriented programming
Jim Anderson, Barry Fishman
Byte, May 1985.

(BAGO85) *Signature Files : an access method for documents*
and its analytical performence evaluation
(ACM Transactions on Office Information Systems
Vol 2, No 4, Oct. 1984 (p. 267-288))
Jean-Marc Bardiaux, Bernard Goffinet
Séminaire : Bureautique (R. Lesuisse)
Institut d'Informatique, 1984-85.

(BDHMP85) *Le langage Smalltalk : programmation par objets*
J-L. Bertrand, B. Daene, J-P Hogne, P. Marechal
D. Penninckx
Séminaire : Méthodologie de programmation (C. Cherton)
Institut d'Informatique, 1984-85.

(BEJO85) *Mémoire : "Réalisation d'un courrier électronique :*
Implémentation sur un réseau local."
Jean-Marie Bernard, Alain Josis
Institut d'Informatique, Sept. 1985.

(BHSL84) *Formanager : an office forms management system*
S. Bing Yao, Alan R. Hevner, Zhougzhi Shi, Dawei Luo
University of Maryland
ACM Transactions on Office Information Systems,
July 1984 (p. 235-262).

(BIM85) *BIM-PROLOG*
User manual
BIM S.A., 1985.

(BMS84) *On conceptual modeling :*
Perspectives form Artificial Intelligence,
Databases, and Programming Languages
M. L. Brodie, J. Mylopoulos, J. W. Schmidt
Springer-Verlag, 1984.

(BOPI83) *Conception assistée des applications informatiques.*
1. Etude d'opportunité et analyse conceptuelle
F. Bodart, Y. Pigneur
Masson, 1983.

BIBLIOGRAPHIE

(BOPI84) *Cours d'analyse fonctionnelle*
F. Bodart, Y. Pigneur
Institut d'Informatique, 1983-84.

(BOUR) *The Unix System*
S R Bourne (Bell Laboratories)
Addison-Wesley Publishing Company.

(BRAC83) *What IS-A is and isn't : an analysis of Taxonomic Links in Semantic Networks*
Ronald J. Brackman
Fairchild Laboratory for Artificial Intelligence Research
Computer, Oct. 1983.

(BYTE84) *Tektronix's 4404 Artificial Intelligence system runs Xerox's Smalltalk-80*
Byte, Oct. 1984.

(CHMU86) *Mémoire : " Contribution à l'atelier logiciel de conception de bases de données : étude des transformations de schémas. "*
Carine Charlot, Irmgard Muller
Institut d'Informatique, Juin 1986.

(DEHO85) *Vitrail : a window manager for an office information system*
(Alain Wegmann : projet Kayak INRIA)
Marc Deroitte, Jean-Pierre Hogne
Séminaire : Bureautique (R. Lesuisse)
Institut d'Informatique, 1984-85.

(DACH86) *Mémoire : " Adaptation du langage DSL aux activités de bureau "*
Nadine Dachouffe
Institut d'informatique, Sept. 1986.

(DAPA85) *Un modèle d'architecture de document du bureau orienté objet*
Nadine Dachouffe, Cécile Paris
Séminaire : Bureautique (R. Lesuisse)
Institut d'Informatique, 1984-85.

(DETI85) *La famille des produits SUN-2 :*
Des souris et des hommes ...
Jean-Marie Deom, Benoit Tilman
Séminaire : Bureautique (R. Lesuisse)
Institut d'Informatique, 1984-85.

(DDMMP85) *Ingres*
L. Delaisse, D. Dewandeleer, C. Mahiat, P. Marchal, B. Pirenne
Séminaire : Technologie des fichiers (J-L. Hainaut)
Institut d'Informatique, 1984-85.

BIBLIOGRAPHIE

(DIGI84) *Methods*
Environment guide and Smalltalk language guide
Digitalk Inc., 1984.

(GAVA86) *Mémoire : " Contribution à la mise en oeuvre d'une couche logiciel permettant un courrier électronique entre réseaux différents. "*
Patrice Gathon, Alain Van Gyseghem
Institut d'Informatique, Sept. 1986.

(GOHE85) *Analyse d'une messagerie vocale*
André Gonay, Benoît Henry
Séminaire : Bureautique (R. Lesuisse)
Institut d'Informatique, 1984-85.

(HAIN81) *Un modèle descriptif de bases de données au niveau organique : le modèle d'accès*
J-L. Hainaut
Institut d'Informatique, 1981.

(HAIN83) *Cadre de référence pour la conception de bases de données*
J-L. Hainaut
Institut d'Informatique, 1983.

(HAIN84A) *Introduction à un système de gestion de bases de données relationnel SQL/DS d'IBM*
J-L. Hainaut
Institut d'Informatique, 1984.

(HAIN84B) *Programmation d'application sur bases de données*
J-L. Hainaut
Institut d'Informatique, 1984.

(HAIN84C) *Cours de conception de fichiers et banques de données*
J-L. Hainaut
Institut d'Informatique, 1983-84.

(HINES85) *Office Automation :*
Tools and Methods for system building
V. Douglas Hines
John Wiley & Sons, 1985.

(HOWI81) *Lisp*
B. Horn, H. Winston
Addison Wesley, 1981.

(KEPO82) *An introduction to programming in C :*
A book on C
Al Kelley, Ira Pohl
The Benjamin/Cummings Publishing Company, 1982.

BIBLIOGRAPHIE

(KERI78) *The C programming language*
Brian W. Kernighan, Dennis M. Ritchie
Bell Laboratories, Murray Hill, New Jersey
Prentice-Hall, 1978.

(LEON86) *Mémoire : " Prise en compte des conditions d'application des normes juridiques dans le cadre d'un système d'aide à la décision judiciaire "*
Sophie Leonard
Institut d'Informatique, Sept. 1986.

(LESU85) *Cours de Bureautique*
R. Lesuisse
Institut d'Informatique, 1984-85.

(LOPI86) *Mémoire : " Interface graphique pour un courrier électronique. "*
Yves Loos, Philippe Piron
Institut d'Informatique, Sept. 1986.

(MACI85) *Smalltalk-PC*
Christopher Macie
Byte, May 1985 (p. 155-165).

(MENO84) *Comprendre les bases de données :*
Théorie et pratique
Alfred Mesguich, Bernard Normier
Masson, 1984.

(MPPRW85) *Laboratoire : "Génération de modules d'accès"*
I. Mathieu, C. Paris, P. Pirson, S. Robert, D. Warnier
Institut d'Informatique, 1984-85.

(NAEL81) *L'archivage électronique*
Albert Nael (CGA-ALCATEL)
Afcet-Sicob Bureautique
Actes du Congrès de Paris 19-22 Mai 1981.

(OLDO85) *Filenet : Document-Image Processor*
System Summary
Olivetti-Dots, 1985.

(RACF85) *Resource Access Control Facility (RACF)*
General information manual
Version 1, release 6.

(REEN81) *User-oriented description of Smalltalk systems*
Trygve M H Reenskaug
Byte, Aug. 1981 (p. 148-168).

BIBLIOGRAPHIE

(ROBE85) *Message files*
(Dennis Tsichristzis, Stravros Christodoulakis
University of Toronto
ACM Transactions on Office Information Systems
Vol. 1 No. 1 Jan. 1983 (p. 88-98))
Stefan Robert
Séminaire : Bureautique (R. Lesuisse)
Institut d'Informatique, 1985-86.

(ROBS81) *Object-oriented software systems*
David Robson
Byte, Aug. 1981 (p. 74-86).

(RUWA85) *Office Automation : a management approach*
Ruprecht, Wagoner
John Wiley & Sons, 1985.

(SIKV82) *Designing the Star User Interface*
D. Canfield Smith, C. Irby, R. Kimball, B. Verplank
Xerox Corporation (Palo Alto)
Byte, Apr. 1982.

(SOBE84) *A practical guide to the Unix System*
Mark G. Sobell
The Benjamin/Cummings Publishing Company, 1984.

(SUN84A) *The Unix System*
A Sun Technical Report
Sun Microsystems, 1984.

(SUN84B) *Sun MicroIngres*
Self-instruction guide
Sun Microsystems, Aug 1984.

(SUN84C) *Sun MicroIngres*
Equel/C user's guide
Sun Microsystems, Aug 1984.

(SUN85) *The Sun-2 Product Family :*
A technical overview
Sun Microsystems, 1985.

(SWK76) *The design and implementation of Ingres*
P. Kreps, M. Stonebraker, E. Wong
University of California, Berkeley
ACM Transaction on Database Systems
Vol 1, No 3, Sept. 1976 (p. 189-222)

(TESL81) *The Smalltalk Environnement*
Larry Tesler
Byte, Aug. 1981 (p. 90-147).

(TSIC85) *Office Automation : Concepts and Tools*
Dionysios Tsichritzis
Springer-Verlag, 1985.

BIBLIOGRAPHIE

(VANL85) *Cours de méthodologie de développment de logiciels*
Alex van Lamsweerde
Institut d'Informatique, 1984-85.

(VANL86) *Cours de méthodologie de développment de logiciels : matière approfondie (Lisp)*
Alex van Lamsweerde
Institut d'Informatique, 1985-86.

(WEBS84) *The Macintosh*
Bruce F. Webster
Byte, Aug 1984 (p. 238-251).

(WEYO85) *Smalltalk comes on the microcomputer world : Methods is object-oriented*
Digitalk Inc. of Los Angeles, California
Bruce Webster, Tom Yonkman
Byte, May 1985 (p. 151-154).

(WOO83) *Authorizations in a message management system*
Carson Woo
University of Toronto
Computer Systems Research Group (1).

(XEROX81) *The Smalltalk-80*
Xerox Learning Research Group
Byte, Aug. 1981 (p. 36-48).

(XEROX85) *Xerox 8000 Network Systems : Star Workstations*
Xerox, 1985.

(1) The Computer Systems Research Group (CSRG) is a inter-disciplinary group formed to conduct research and development relevant to computer systems and their applications. It is joinly administered by the Department of Electrical Engineering and the Department of Computer Science of the University of Toronton, and is supported in part by the Natural Sciences and Engineering Research Council of Canada.

Facultés Universitaires Notre-Dame de la Paix à Namur

Institut d'informatique

Annexes :

Analyse et implémentation d'un environnement de rangement d'informations.

Mémoire présenté par
Stefan ROBERT
en vue de l'obtention du titre de
Licencié et Maître en Informatique

Promoteur : R. Lesuisse

Année académique 1985-86.

ANNEXES

PLAN DES ANNEXES : Pages

ANNEXE A : SOURCES DES PROCEDURES DE GESTION DES OBJETS DE L'E.R.

INSTAL.QC	3
ETAT.QC	7
INSTAL.DOC	14
LIB_C.QC	16
LIB_V.QC	23
LIB_I.QC	24
LIB_F.QC	44
LIB_T.QC	51
CREAT.QC	58
SUPPR.QC	64
MODIF.QC	68
CONSUL.QC	74
MAJ.QC	77
CONTEN.QC	80
RECH.QC	87
CREATION.C	92
SUPPRESSION.C	100
MODIFICATION.C	105
CONSULTATION.C	112
MISE_A_JOUR.C	116
RECHERCHE.C	120
PROPRIETE.C	124

ANNEXES

		Pages
ANNEXE B :	SOURCES DES PROGRAMMES DE TEST	
	MAKEFILE	3
	TCR.C	6
	TCR.OUT	26
	TCR.DOC	27
	TSUP.C	37
	TSUP.OUT	41
	TSUP.DOC	42
	TMO.C	52
	TMO.OUT	62
	TMO.DOC	63
	TCOMA.C	67
	TCOMA.OUT	76
	TCOMA.DOC	77
	TRE.C	80
	TRE.OUT	93
	TPR.C	96
	TPR.OUT	101
	TCONTEN.C	105
	TCONTEN.OUT	108
ANNEXE C :	SOURCES DE FONCTIONS LISP DE GENERATION AUTOMATIQUE DE REQUETES INGRES	
	START.LSP	2
	FORM1.LSP	2
	FORM2.LSP	3
	GENERE.LSP	3

ANNEXE A : SOURCES DES PROCEDURES DE GESTION DES OBJETS DE L'E.R.

L'annexe A contient l'ensemble des programmes et procédures qui permettent la gestion des objets de l'E.R. introduits au chapitre 3.

La notation des extensions des fichiers suivra la convention suivante :

- les fichiers notés 'QC' contiennent des instructions en langage C ainsi que des requêtes Ingres (C/EQUEL) ;
- les fichiers notés 'C' contiennent uniquement des instructions en langage C ;
- le fichier noté 'DOC' contient le texte généré par le programme ETAT. Ce programme est un des outils qui a servi lors des tests. Il permet de sortir l'état de l'E.R. à un moment donné c'est-à-dire, par type d'objet, toutes les occurrences d'objets existant dans l'E.R. ainsi que leurs propriétés.

Les fichiers correspondant accompagnés d'une note explicative de leur contenu sont :

INSTAL.QC : programme INSTAL permettant de créer l'E.R. et d'y adjoindre les premiers objets nécessaires à la gestion de l'E.R. c'est-à-dire l'administrateur de l'E.R. ainsi que son groupe, l'E.R. et la poubelle.

ETAT.QC : programme ETAT introduit ci-dessus.

INSTAL.DOC : résultat de ETAT après l'installation de l'E.R.

LIB_C.QC : l'ensemble des constantes et structures de données utilisées par les autres procédures.

LIB_V.QC : les variables globales utilisées par l'ensemble des procédures.

LIB_I.QC : les fonctions de base faisant appel à des requêtes Ingres, utilisées par certaine procédures.

LIB_F.C : les fonctions de base, en C, utilisées par quelques procédures.

LIB_T.C : les fonctions de base, en C, utilisées par les programmes de test décrits dans l'annexe B ainsi que INSTAL.

CREAT.QC --> RECH.QC : les opérations proprement dites sur les objets de l'E.R. au niveau de l'SGBD cible choisi (INGRES).

CREATION.C --> PROPRIETE.C : les opérations directement accessibles à une application. Elles vérifient un certain nombre de conditions avant de faire appel à l'opération proprement dite sur les objets de l'E.R. Elles sont, contrairement au groupe précédent, indépendantes de l'SGBD choisi pour l'implémentation de l'E.R.

```
#  include "stdio.h"
## include "lib_c.qc"
## include "lib_v.qc"


/*****************************************************************
*****************************************************************
**                    PROGRAMME D'INSTALLATION                   **
**          DE L'ENVIRONNEMENT DE RANGEMENT (E.R.)               **
*****************************************************************
*****************************************************************/

/* Specification : ce programme construit la configuration
   minimale le l'E.R. pour qu'il puisse par la suite etre
   exploite.

   Cette configuration se compose :

   - des differents types d'objets qui seront utilises
     (utilisateur, groupe_utilisateur, env_rang, armoire,
     tiroir, farde, poubelle, boite, groupe_information
     et information) ;

   - la table 'compteur' pour la generation automatique
     de codes ;

   - l'utilisateur particulier 'administrateur' ;

   - le groupe_utilisateur de l'administrateur ;

   - l'env_rang ;

   - la poubelle ;
*/

main ()

{
/* DEBUT DU PROGRAMME PRINCIPAL */

/* initialisation des variables qui servent de constantes pour
   l'interface Ingres : */

   cadm   = 1 ;
   cenv   = 2 ;
   cpou   = 3 ;
   cgradm = 4 ;

   printf ("Installation E.R. ...\n") ;

/* creation de la base de donnees */
   system ("createdb er > .#bidon") ;   /* appel operation Unix */

/* ouverture de la base de donnees */
## ingres "er"

## create utilisateur (nom=c25,code=i4,
```

```
##                        pwd=c15,crdate=date,
##                        creat=i4,perm=c1,
##                        groupe=c20)

## create gr_util   (nom=c25,code=i4,
##                   crdate=date,creat=i4)

## create env_rang   (nom=c25,code=i4,
##                    crdate=date,creat=i4,
##                    stconf=c1,poss=c50,
##                    desc=c50)

## create armoire   (nom=c25,code=i4,
##                   crdate=date,creat=i4,
##                   stconf=c1,poss=c50,
##                   type=c15,desc=c50)

## create tiroir    (nom=c25,code=i4,
##                   crdate=date,creat=i4,
##                   stconf=c1,poss=c50,
##                   type=c15,desc=c50,
##                   code_armoire=i4)

## create farde   (nom=c25,code=i4,
##                 crdate=date,creat=i4,
##                 stconf=c1,poss=c50,
##                 type=c15,desc=c50,
##                 code_tiroir=i4)

## create poubelle   (nom=c25,code=i4,
##                    crdate=date,creat=i4,
##                    stconf=c1,poss=c50,
##                    desc=c50)

## create boite   (nom=c25,code=i4,
##                 crdate=date,creat=i4,
##                 stconf=c1,poss=c50,
##                 type=c15,desc=c50)

## create gr_info   (nom=c25,code=i4,
##                   crdate=date,creat=i4,
##                   presence=i2,stconf=c1,
##                   poss_con=c50,poss_maj=c50,
##                   type=c15,desc=c50,
##                   cont_t=i2,cont_c=i4)

## create information   (nom=c25,code=i4,
##                       presence=i2,nature=c1,
##                       crdate=date,creat=i4,
##                       codate=date,consult=i4,
##                       modate=date,modificat=i4,
##                       stconf=c1,
##                       poss_con=c50,poss_maj=c50,
##                       type=c15,desc=c50,
##                       exped=c50,mots_cles=c50,
##                       ref_stck=c15,ref_orig=c15,
##                       cont_t=i2,cont_c=i4)
```

```
## create compteur (code=i4)

/* declaration des abreviations */
## range of u  is utilisateur
## range of gu is gr_util
## range of e  is env_rang
## range of a  is armoire
## range of t  is tiroir
## range of f  is farde
## range of p  is poubelle
## range of b  is boite
## range of gi is gr_info
## range of i  is information
## range of c  is compteur


/* organisation des tables sur une cle pour ameliorer
   la performance d'acces aux objets. */

## modify utilisateur to hash on code

## modify gr_util to hash on nom

## modify armoire to hash on code

## modify tiroir to hash on code

## modify farde to hash on code

## modify boite to hash on code

## modify gr_info to hash on code

## modify information to hash on code


/* creation d'index secondaires pour ameliorer
   la performance des acces aux objets */

## index on utilisateur is u_index (groupe)

## index on tiroir is t_index (code_armoire)

## index on farde is f_index (code_tiroir)

## index on gr_info is gi_index (cont_t, cont_c)

## index on information is i_index (cont_t, cont_c)


/* creation du groupe_utilisateur de l'administrateur */
## append gr_util  (nom="/",
##                  code=cgradm,
##                  crdate="01-jan-86",
##                  creat=cadm)
```

```
/* creation de l'utilisateur administrateur */
## append to utilisateur (nom="ADMINISTRATEUR",
##                        code=cadm,
##                        pwd="code_ADM",
##                        crdate= "01-jan-86",
##                        creat=cadm,
##                        perm="E",
##                        groupe="/" )

/* creation de l'env_rang */
## append to env_rang    (nom="ENVIRONNEMENT RANGEMENT",
##                        code=cenv,
##                        crdate= "01-jan-86",
##                        creat=cadm,
##                        stconf="N",
##                        poss="*",
##                        desc="description env_rang")

/* creation de la poubelle */
## append to poubelle    (nom="POUBELLE",
##                        code=cpou,
##                        crdate= "01-jan-86",
##                        creat=cadm,
##                        stconf="N",
##                        poss="*",
##                        desc="description poubelle")

/* initialisation du premier code genere */
## append to compteur    (code=4)

printf ("Installation ok.\n") ;

}  /* FIN DU PROGRAMME INSTAL */
```

```
#  include "stdio.h"
## include "lib_c.qc"
## include "lib_v.qc"


/*****************************************************************
*****************************************************************
**                        PROGRAMME   ETAT                     **
*****************************************************************
*****************************************************************/

/* Specification : ce programmme permet d'afficher le contenu
   de l'environnement de rangement par categorie d'objets :
   utilisateurs, groupes d'utilisateurs , ...
   Pour chacun des objets on peut voir la valeur de chacune
   de leurs proprietes.

   Il servira comme programme de test des differents modules
   pour observer l' 'etat'  de l'E.R. avant et apres execution
   du module proprement dit.
*/

main ()

{
/* DEBUT DU PROGRAMME PRINCIPAL */

/* initialisation des variables qui servent de constantes pour
   l'interface Ingres : */

   cadm    = 1 ;
   cenv    = 2 ;
   cpou    = 3 ;
   cgradm = 4 ;

/* Ouverture de la base de donnees ---------------------- */
   ouv_bd () ;

   etat_util () ;
   etat_gr_uti () ;
   etat_env_rang () ;
   etat_arm () ;
   etat_tir () ;
   etat_far () ;
   etat_pou () ;
   etat_boi () ;
   etat_gr_inf () ;
   etat_inf () ;

/* Fermeture de la base de donnees ---------------------- */
   ferm_bd () ;

}  /* FIN DU PROGRAMME PRINCIPAL */
```

```
/*************************************************************
*                    PROCEDURE  ETAT_UTIL                    *
*************************************************************/

etat_util ()

{
/* DECLARATION DES VARIABLES LOCALES A LA PROCEDURE */
## struct ituti uobj ;
   int     num = 0 ;          /* compteur pour numerotation
                                 des objets */

/* DEBUT DE LA PROCEDURE ETAT_UTIL */

   printf ("UTILISATEUR(S) :\n") ;
   printf ("===============\n \n") ;

## retrieve (uobj.ucode=u.code,uobj.unom=u.nom,
##           uobj.upwd=u.pwd,uobj.ucrdate=u.crdate,
##           uobj.ucreat=u.creat,uobj.uperm=u.perm,
##           uobj.ugroupe=u.groupe)
## {
   pr_uti (uobj,++num) ;
## }

   return (0) ;

}  /* FIN DE LA PROCEDURE ETAT_UTIL */


/*************************************************************
*                  PROCEDURE  ETAT_GR_UTI                    *
*************************************************************/

etat_gr_uti ()

{
/* DECLARATION DES VARIABLES LOCALES A LA PROCEDURE */
## struct itgruti guobj ;
   int     num = 0 ;          /* compteur pour numerotation
                                 des objets */

/* DEBUT DE LA PROCEDURE ETAT_GR_UTI */

   printf ("GROUPE(S) UTILISATEURS :\n") ;
   printf ("========================\n \n") ;

## retrieve (guobj.gucode=gu.code,guobj.gunom=gu.nom,
##           guobj.gucrdate=gu.crdate,guobj.gucreat=gu.creat)
## {
   pr_gr_uti (guobj,++num) ;
## }

   return (0) ;

}  /* FIN DE LA PROCEDURE ETAT_GR_UTIL */
```

```
/*****************************************************************
*                  PROCEDURE  ETAT_ENV_RANG                      *
*****************************************************************/

etat_env_rang ()

{
/* DECLARATION DES VARIABLES LOCALES A LA PROCEDURE */
## struct itenvran eobj ;
   int     num = 0 ;          /* compteur pour numerotation
                                 des objets */

/* DEBUT DE LA PROCEDURE ETAT_ENV_RANG */

   printf ("ENVIRONNEMENT DE RANGEMENT :\n") ;
   printf ("==========================\n \n") ;

## retrieve (eobj.ecode=e.code,eobj.enom=e.nom,
##           eobj.ecrdate=e.crdate,eobj.ecreat=e.creat,
##           eobj.edesc=e.desc,eobj.estconf=e.stconf,
##           eobj.eposs=e.poss)
## {
   pr_env_ran (eobj,++num) ;
## }

   return (0) ;

}  /* FIN DE LA PROCEDURE ETAT_ENV_RANG */


/*****************************************************************
*                  PROCEDURE  ETAT_ARM                           *
*****************************************************************/

etat_arm ()

{
/* DECLARATION DES VARIABLES LOCALES A LA PROCEDURE */
## struct itarm aobj ;
   int     num = 0 ;          /* compteur pour numerotation
                                 des objets */

/* DEBUT DE LA PROCEDURE ETAT_ARM */

   printf ("ARMOIRE(S) :\n") ;
   printf ("==========\n \n") ;

## retrieve (aobj.acode=a.code,aobj.anom=a.nom,
##           aobj.acrdate=a.crdate,aobj.acreat=a.creat,
##           aobj.adesc=a.desc,aobj.astconf=a.stconf,
##           aobj.atype=a.type,aobj.aposs=a.poss)
## {
   pr_arm (aobj,++num) ;
## }
```

```
   return (0) ;

}  /* FIN DE LA PROCEDURE ETAT_ARM */


/*****************************************************************
*                  PROCEDURE  ETAT_TIR                           *
*****************************************************************/

etat_tir ()

{
/* DECLARATION DES VARIABLES LOCALES A LA PROCEDURE */
## struct ittir tobj ;
   int     num = 0 ;          /* compteur pour numerotation
                                 des objets */

/* DEBUT DE LA PROCEDURE ETAT_TIR */

   printf ("TIROIR(S) :\n") ;
   printf ("==========\n \n") ;

## retrieve (tobj.tcode=t.code,tobj.tnom=t.nom,
##           tobj.tcrdate=t.crdate,tobj.tcreat=t.creat,
##           tobj.tdesc=t.desc,tobj.tstconf=t.stconf,
##           tobj.ttype=t.type,tobj.tposs=t.poss,
##           tobj.tcont_c=t.code_armoire)
## {
   pr_tir (tobj,++num) ;
## }

   return (0) ;

}  /* FIN DE LA PROCEDURE ETAT_TIR */


/*****************************************************************
*                  PROCEDURE  ETAT_FAR                           *
*****************************************************************/

etat_far ()

{
/* DECLARATION DES VARIABLES LOCALES A LA PROCEDURE */
## struct itfar fobj ;
   int     num = 0 ;          /* compteur pour numerotation
                                 des objets */

/* DEBUT DE LA PROCEDURE ETAT_FAR */

   printf ("FARDE(S) :\n") ;
   printf ("=========\n \n") ;

## retrieve (fobj.fcode=f.code,fobj.fnom=f.nom,
##           fobj.fcrdate=f.crdate,fobj.fcreat=f.creat,
##           fobj.fdesc=f.desc,fobj.fstconf=f.stconf,
##           fobj.ftype=f.type,fobj.fposs=f.poss,
```

```
##            fobj.fcont_c=f.code_tiroir)
## {
   pr_far (fobj,++num) ;
## }

   return (0) ;

}  /* FIN DE LA PROCEDURE ETAT_FAR */


/*************************************************************
*                  PROCEDURE  ETAT_POU                       *
*************************************************************/

etat_pou ()

{
/* DECLARATION DES VARIABLES LOCALES A LA PROCEDURE */
## struct itpou pobj ;
   int     num = 0 ;          /* compteur pour numerotation
                                 des objets */

/* DEBUT DE LA PROCEDURE ETAT_POU */

   printf ("POUBELLE :\n") ;
   printf ("========\n \n") ;

## retrieve (pobj.pcode=p.code,pobj.pnom=p.nom,
##           pobj.pcrdate=p.crdate,pobj.pcreat=p.creat,
##           pobj.pdesc=p.desc,pobj.pstconf=p.stconf,
##           pobj.pposs=p.poss)
## {
   pr_pou (pobj,++num) ;
## }

   return (0) ;

}  /* FIN DE LA PROCEDURE ETAT_POU */


/*************************************************************
*                  PROCEDURE  ETAT_BOI                       *
*************************************************************/

etat_boi ()

{
/* DECLARATION DES VARIABLES LOCALES A LA PROCEDURE */
## struct itboi bobj ;
   int     num = 0 ;          /* compteur pour numerotation
                                 des objets */

/* DEBUT DE LA PROCEDURE ETAT_FAR */

   printf ("BOITE(S) :\n") ;
   printf ("========\n \n") ;
```

```
## retrieve (bobj.bcode=b.code,bobj.bnom=b.nom,
##           bobj.bcrdate=b.crdate,bobj.bcreat=b.creat,
##           bobj.bdesc=b.desc,bobj.bstconf=b.stconf,
##           bobj.btype=b.type,bobj.bposs=b.poss)
## {
   pr_boi (bobj,++num) ;
## }

   return (0) ;

}  /* FIN DE LA PROCEDURE ETAT_BOI */


/*****************************************************************
*                PROCEDURE  ETAT_GR_INF                          *
*****************************************************************/

etat_gr_inf ()

{
/* DECLARATION DES VARIABLES LOCALES A LA PROCEDURE */
## struct itgrinf giobj ;
   int    num = 0 ;          /* compteur pour numerotation
                                des objets */

/* DEBUT DE LA PROCEDURE ETAT_GR_INF */

   printf ("GROUPE(S) INFORMATIONS :\n") ;
   printf ("=======================\n \n") ;

## retrieve (giobj.gicode=gi.code,giobj.ginom=gi.nom,
##           giobj.gicrdate=gi.crdate,giobj.gicreat=gi.creat,
##           giobj.gipresence=gi.presence,
##           giobj.gidesc=gi.desc,giobj.gistconf=gi.stconf,
##           giobj.gitype=gi.type,
##           giobj.giposs_con=gi.poss_con,
##           giobj.giposs_maj=gi.poss_maj,
##           giobj.gicont_t=gi.cont_t,
##           giobj.gicont_c=gi.cont_c)
## {
   pr_gr_inf (giobj,++num) ;
## }

   return (0) ;

}  /* FIN DE LA PROCEDURE ETAT_GR_INF */


/*****************************************************************
*                PROCEDURE  ETAT_INF                             *
*****************************************************************/

etat_inf ()

{
/* DECLARATION DES VARIABLES LOCALES A LA PROCEDURE */
## struct itinf iobj ;
```

```
   int     num = 0 ;          /* compteur pour numerotation
                                 des objets */

/* DEBUT DE LA PROCEDURE ETAT_FAR */

   printf ("INFORMATION(S) :\n") ;
   printf ("==============\n \n") ;

## retrieve (iobj.icode=i.code,iobj.inom=i.nom,
##            iobj.inature=i.nature,iobj.ipresence=i.presence,
##            iobj.icrdate=i.crdate,iobj.icreat=i.creat,
##            iobj.icodate=i.codate,iobj.iconsult=i.consult,
##            iobj.imodate=i.modate,iobj.imodificat=i.modificat,
##            iobj.idesc=i.desc,iobj.istconf=i.stconf,
##            iobj.itype=i.type,iobj.iposs_con=i.poss_con,
##            iobj.iposs_maj=i.poss_maj,iobj.iexped=i.exped,
##            iobj.imots_cles=i.mots_cles,
##            iobj.iref_stck=i.ref_stck,
##            iobj.iref_orig=i.ref_orig,iobj.icont_t=i.cont_t,
##            iobj.icont_c=i.cont_c)
## (
   pr_inf (iobj,++num) ;
## )

   return (0) ;

}  /* FIN DE LA PROCEDURE ETAT_INFORMATION */
```

```
UTILISATEUR(S) :
================

1.
      nom     : ADMINISTRATEUR
      code    : 1
      crdate :  1-jan-1986
      creat   : 1
      perm    : E
      groupe : /

GROUPE(S) UTILISATEURS :
========================

1.
      nom     : /
      code    : 4
      crdate :  1-jan-1986
      creat   : 1

ENVIRONNEMENT DE RANGEMENT :
============================

1.
      nom     : ENVIRONNEMENT RANGEMENT
      code    : 2
      crdate :  1-jan-1986
      creat   : 1
      desc    : description env_rang
      stconf : N
      poss    : *

ARMOIRE(S) :
============

TIROIR(S) :
===========

FARDE(S) :
==========

POUBELLE :
==========

1.
      nom       : POUBELLE
      code      : 3
      crdate    :  1-jan-1986
      creat     : 1
      desc      : description poubelle
      stconf    : N
      poss      : *

BOITE(S) :
==========
```

GROUPE(S) INFORMATIONS :
==========

INFORMATION(S) :
==========

```
/*************************************************************
*      CONSTANTES  DE  TYPE  1  :  LIEES  AUX  OBJETS       *
*************************************************************/

/* CONSTANTES UTILISEES POUR L'IDENTIFICATION DES TYPE D'OBJETS */

# define sutil       1      /* valeur du switch sur UTILISATEUR */
# define sgr_util    2      /* valeur du switch sur GR_UTIL */
# define senv_rang   3      /* valeur du switch sur ENV_RANG */
# define sarm        4      /* valeur du switch sur ARMOIRE */
# define stir        5      /* valeur du switch sur TIROIR */
# define sfar        6      /* valeur du switch sur FARDE */
# define spou        7      /* valeur du switch sur POUBELLE */
# define sboi        8      /* valeur du switch sur BOITE */
# define sgr_info    9      /* valeur du switch sur GR_INFO */
# define sinfo       10     /* valeur du switch sur INFORMATION */

/*************************************************************
 Remarque : pour les types chaine de caracteres, la valeur
            de LGMAX*** sera egale a la longueur du champ
            correspondant de la zone INGRES + 1
            le caractere supplementaire servant au \0 en C.
 *************************************************************/

/* VALEURS REPRESENTANT LES LONGUEURS MAXIMUM DES ATTRIBUTS */

# define  LGMAXTOB  1      /* long max du type type tobject */
# define  LGMAXNOM  26     /* long max du type nom */
# define  LGMAXPWD  16     /* long max du type pwd */
# define  LGMAXGRO  21     /* long max du type groupe */
# define  LGMAXPOS  51     /* long max du type poss */
# define  LGMAXDES  51     /* long max du type desc */
# define  LGMAXTYP  16     /* long max du type type */
# define  LGMAXSTC  2      /* long max du type stconf */
# define  LGMAXPER  2      /* long max du type perm */
# define  LGMAXEXP  51     /* long max du type expedition */
# define  LGMAXMCL  51     /* long max du type mots_cles */
# define  LGMAXNAT  2      /* long max du type nature */
# define  LGMAXSTK  16     /* long max du type ref_stck */
# define  LGMAXORI  16     /* long max du type ref_orig */


/* VALEURS POSSIBLES DE LA NATURE D'UNE INFORMATION */

# define  DOCUMENT    'D'  /* valeur attribut nature */
# define  MESSAGE     'M'  /* valeur attribut nature */
# define  FORMULAIRE  'F'  /* valeur attribut nature */

/* VALEURS POSSIBLES DE LA PERMISSION D'UN UTILISATEUR */

# define  LECTURE    'L'   /* valeur possible de l'attr. perm */
# define  ECRITURE   'E'   /* valeur possible de l'attr. perm */

/* VALEURS POSSIBLES DU MODE D'OUVERTURE D'UNE CONSULTATION
   OU D'UNE MISE A JOUR   */

# define  OUVERTURE  'O'   /* mode consultation, mise_a_jour */
```

```
# define  FERMETURE 'F'    /* mode consultation, mise_a_jour */

/* VALEURS POSSIBLES DU STATUT DE CONFIDENTIALITE
   D'UNE INFORMATION */

# define  CONFIDENTIEL      'C' /* valeur possible de  stconf */
# define  NON_CONFIDENTIEL 'N' /* valeur possible de  stconf */

/* VALEURS POSSIBLES DE LA PRESENCE D'UNE INFORMATION */

# define  MAXPRES   32500    /* valeur max de presence */

# define  PRESENT     0       /* val. poss. de presence */
# define  MAJ      MAXPRES    /* val. poss. de presence = MAJ*/
                          /* val. 1 a MAXPRES -1 = CONSULTATION */


          /*******************************************
          *  VALEURS POSSIBLES DE CODE_RET, CODE_ERR  *
          *******************************************/

/* Tout c'est bien passe, pas d'erreur detectee */

# define  OK              0    /* val. poss. de code_ret, code_err */


/* Erreurs sur l'attribut presence */

# define  ERR_PRES     101   /* depassement nb max. consultations */
# define  CONT_PRES    102   /* depassement nb min. consultations */


/* Erreurs dans la procedure gen_compt () */

# define  GEN_CODE_R  201   /* erreur lors de la lecture du code */
# define  GEN_CODE_W  202   /* erreur lors de l'ecriture du code */
# define  GEN_CODE_P  203   /* contrainte d'integrite violee :
                               plusieurs compteurs */


/* Erreurs de non correspondance aux specifications */

# define  NOT_IMPLEM  301   /* operation non-implementee sur cet
                               objet  */
# define  OBJ_INDEF   311   /* type objet indefini ou pas
                               d'application de cette operation
                               sur ce type d'objet  */
# define  OP_TOB      312    /* cette operation n'est pas
                                executable sur ce type d'objet */
# define  TCONT_INDEF  321   /* type objet contenant indefini */
# define  CONT_INEXIST 322   /* code objet contenant inexistant */
# define  MODE_INEXIST 331   /* mode de consultation, mise a jour
                               inexistant */


/* Erreurs lors de la validation des parametres des operations */
```

```
/* Problemes d'acces aux objets */

# define   TOB_AC        401   /* type d'objet non accedable */
# define   AC            402   /* possibilite d'acces a l'objet */
# define   AC_ADM        403   /* acces administrateur uniquement */
# define   AC_CREAT      404   /* acces createur uniquement :
                                  obj. confid... */
# define   AC_ECRIT      405   /* permission insuffisante de
                                  l'utilisateur */

# define   AC_SUPP       411   /* probleme lors de l'acces pour
                                  suppression : utilisateur non connu
                                  ou non proprietaire de l'objet */

# define   AC_CONSUL     412   /* probleme lors de l'acces pour
                                  consultation : possibilite d'acces
                                  en consultation a l'objet */
# define   AC_MAJ        413   /* probleme lors de l'acces pour mise
                                  a jour : possibilite d'acces en
                                  mise_a_jour a l'objet */
# define   AC_MODIF      414   /* probleme lors de l'acces pour
                                  modification : utilisateur non
                                  connu ou non proprietaire
                                  de l'objet */

# define   DISP_OBJ      421   /* probleme de disponibilite de
                                  l'objet pour consultation ou mise
                                  a jour */
# define   OBJ_NON_VIDE  422   /* objet non vide, ne peut etre
                                  supprime */
# define   INF_NON_PRES  423   /* information non presente dans
                                  l'E.R.


/* Problemes lors des acces physiques aux objets :
   dans ce cas lors d'operation Ingres.  */

# define   BD_CONT_INT   501   /* plusieurs objets de meme code */
# define   BD_O_INEXIST  502   /* pas d'objet de ce code */
# define   BD_I_INEXIST  503   /* pas d'identification possible
                                 pas d'objet de cette identification */

# define   BD_O_CREAT    511   /* probleme lors de la creation de
                                  l'obj. */
# define   BD_O_SUPP     521   /* probleme lors de la suppression de
                                  l'objet
                                  ou pas d'objet de ce code */
# define   BD_OU_CONS    531   /* probleme lors de la consultation de
                                  l'objet en ouverture ou pas d'objet
                                  de ce code */
# define   BD_OF_CONS    532   /* probleme lors de la consultation de
                                  l'objet en fermeture ou pas d'objet
                                  de ce code */
# define   BD_OU_MAJ     541   /* probleme lors de la mise a jour de
                                  l'objet en ouverture ou pas d'objet
                                  de ce code */
# define   BD_OF_MAJ     542   /* probleme lors de la mise a jour de
```

```
                                    l'objet en fermeture ou pas d'objet
                                    de ce code */
# define  BD_O_MODIF    551   /* aucune modification n'a ete operee:
                                    proprietes identiques, pas d'objet
                                    de ce code ou probleme Ingres */


    /*************************************************************
    *     DECLARATION DES STRUCTURES DE DONNEES UTILISEES       *
    *************************************************************/

struct user                                 /* utilisateur */
  { long cod ;                              /* code utilisateur */
    char per ;                              /* permission utilisateur */
    char group[LGMAXGRO] ;                  /* groupe utilisateurs */
  } ;


struct l
  {  long      val ;                        /* valeur */
     struct l *suiv ;                       /* adresse suivant */
  } ;

typedef struct l list ;      /* liste de codes pour recherche () */


/* TYPES D'OBJETS AVEC LEURS PROPRIETES DISTINCTES */

## struct ituti                             /* UTILISATEUR */
##  {  char  unom[LGMAXNOM] ;               /* nom */
##     long  ucode ;                        /* code */
##     char  upwd[LGMAXPWD] ;               /* mot de passe */
##     char  ucrdate[26] ;                  /* date de creation */
##     long  ucreat ;                       /* createur */
##     char  uperm[LGMAXPER] ;              /* permission de l'util. */
##     char  ugroupe[LGMAXGRO] ;            /* groupe de l'utilisateur */
##  } ;

## struct itgruti                           /* GROUPE_UTILISATEUR */
##   {  char  gunom[LGMAXNOM] ;             /* nom */
##      long  gucode ;                      /* code */
##      char  gucrdate[26] ;                /* date de creation */
##      long  gucreat ;                     /* createur */
##   } ;

## struct itenvran                          /* ENV_RANG */
##   {  char  enom[LGMAXNOM] ;              /* nom */
##      long  ecode ;                       /* code */
##      char  ecrdate[26] ;                 /* date de creation */
##      long  ecreat ;                      /* createur */
##      char  edesc[LGMAXDES] ;             /* description */
##      char  estconf[LGMAXSTC] ;           /* statut de confidentialite*/
##      char  eposs[LGMAXPOS] ;             /* possibilite d'acces */
##   } ;

## struct itarm                             /* ARMOIRE */
```

```
##    {  char  anom[LGMAXNOM] ;        /* nom */
##       long  acode ;                 /* code */
##       char  acrdate[26] ;           /* date de creation */
##       long  acreat ;                /* createur */
##       char  adesc[LGMAXDES] ;       /* description */
##       char  astconf[LGMAXSTC] ;     /* statut de confidentialite*/
##       char  atype[LGMAXTYP] ;       /* type */
##       char  aposs[LGMAXPOS] ;       /* possibilite d'acces */
##    } ;

## struct ittir                        /* TIROIR */
##    {  char  tnom[LGMAXNOM] ;        /* nom */
##       long  tcode ;                 /* code */
##       char  tcrdate[26] ;           /* date de creation */
##       long  tcreat ;                /* createur */
##       char  tdesc[LGMAXDES] ;       /* description */
##       char  tstconf[LGMAXSTC] ;     /* statut de confidentialite*/
##       char  ttype[LGMAXTYP] ;       /* type */
##       char  tposs[LGMAXPOS] ;       /* possibilite d'acces */
##       long  tcont_c ;               /* code du contenant */
##    } ;

## struct itfar                        /* FARDE */
##    {  char  fnom[LGMAXNOM] ;        /* nom */
##       long  fcode ;                 /* code */
##       char  fcrdate[26] ;           /* date de creation */
##       long  fcreat ;                /* createur */
##       char  fdesc[LGMAXDES] ;       /* description */
##       char  fstconf[LGMAXSTC] ;     /* statut de confidentialite*/
##       char  ftype[LGMAXTYP] ;       /* type */
##       char  fposs[LGMAXPOS] ;       /* possibilite d'acces */
##       long  fcont_c ;               /* code du contenant */
##    } ;

## struct itpou                        /* POUBELLE */
##    {  char  pnom[LGMAXNOM] ;        /* nom */
##       long  pcode ;                 /* code */
##       char  pcrdate[26] ;           /* date de creation */
##       long  pcreat ;                /* createur */
##       char  pdesc[LGMAXDES] ;       /* description */
##       char  pstconf[LGMAXSTC] ;     /* statut de confidentialite*/
##       char  pposs[LGMAXPOS] ;       /* possibilite d'acces */
##    } ;

## struct itboi                        /* BOITE */
##    {  char  bnom[LGMAXNOM] ;        /* nom */
##       long  bcode ;                 /* code */
##       char  bcrdate[26] ;           /* date de creation */
##       long  bcreat ;                /* createur */
##       char  bdesc[LGMAXDES] ;       /* description */
##       char  bstconf[LGMAXSTC] ;     /* statut de confidentialite*/
##       char  btype[LGMAXTYP] ;       /* type */
##       char  bposs[LGMAXPOS] ;       /* possibilite d'acces */
##    } ;

## struct itgrinf                      /* GROUPE_INFORMATION */
##    {  char  ginom[LGMAXNOM] ;       /* nom */
```

```
##       long  gicode ;                  /* code */
##       char  gicrdate[26] ;            /* date de creation */
##       long  gicreat ;                 /* createur */
##       int   gipresence ;              /* presence */
##       char  gidesc[LGMAXDES] ;        /* description */
##       char  gistconf[LGMAXSTC] ;      /* statut de confidentialite*/
##       char  gitype[LGMAXTYP] ;        /* type */
##       char  giposs_con[LGMAXPOS] ;/* possibilite d'acces en
                                           consultation*/
##       char  giposs_maj[LGMAXPOS] ;/* possibilite d'acces en mise
                                           a jour */
##       int   gicont_t ;                /* type du contenant */
##       long  gicont_c ;                /* code du contenant */
##    } ;

## struct itinf                         /* INFORMATION */
##    {  char  inom[LGMAXNOM] ;          /* nom */
##       long  icode ;                   /* code */
##       char  inature[LGMAXNAT] ;       /* nature de l'information */
##       char  icrdate[26] ;             /* date de creation */
##       long  icreat ;                  /* createur */
##       int   ipresence ;               /* presence */
##       char  icodate[26] ;             /* date de derniere
                                           consultation */
##       long  iconsult ;                /* utilisateur qui a consulte
                                           pour la derniere fois */
##       char  imodate[26] ;             /* date de derniere
                                           modification */
##       long  imodificat ;              /* utilisateur qui a modifie
                                           pour la derniere fois */
##       char  idesc[LGMAXDES] ;         /* description */
##       char  istconf[LGMAXSTC] ;       /* statut de confidentialite*/
##       char  itype[LGMAXTYP] ;         /* type */
##       char  iposs_con[LGMAXPOS] ;     /* possibilite d'acces en
                                           consultation*/
##       char  iposs_maj[LGMAXPOS] ;     /* possibilite d'acces en mise
                                           a jour */
##       char  iexped[LGMAXEXP] ;        /* expedition */
##       char  imots_cles[LGMAXMCL] ;/* mots-cles */
##       char  iref_stck[LGMAXSTK] ;     /* reference de stockage */
##       char  iref_orig[LGMAXORI] ;     /* reference de l'original */
##       int   icont_t ;                 /* type du contenant */
##       long  icont_c ;                 /* code du contenant */
##    } ;

union object
  { struct ituti u ;
    struct itgruti gu ;
    struct itenvran e ;
    struct itarm a ;
    struct ittir t ;
    struct itfar f ;
    struct itpou p ;
    struct itboi b ;
    struct itgrinf gi ;
    struct itinf i ;
  } ;
```

```
/* DECLARATION DU TYPE D'OBJET ABSTRAIT QUI POURRA CONTENIR
   N'IMPORTE QUEL TYPE D'OBJET DEFINI PLUS HAUT */

typedef struct
  {  int type_object ;
     union object o ;
  } tobject ;
```

```
/*****************************************************************
*          DECLARATION  DES  VARIABLES  GLOBALES                 *
*****************************************************************/

##  char cuposs[LGMAXPOS] ; /* copie de uposs pour ingres :
                           acces (), acces_con (), acces_maj () */

##  int  row_count ;    /* gestion d'erreur ingres : nombre de
                           lignes affectees par la requete */
/* les variables declarees en Ingres sont considerees
   comme globales pour lui ; les declarer deux fois produit
   l'erreur 'redeclared' */

##  int  erreur_num ;   /* gestion d'erreur ingres : numero
                           de l'erreur */

/* Solution pour pouvoir utiliser des constantes dans des requetes
   Ingres : utiliser des variables */

## int cadm   ;        /* valeur de l'attr. code de l'util. adm */
## int cenv   ;        /* valeur de l'attr. code de l'env_rang */
## int cpou   ;        /* valeur de l'attr. code de la poubelle */
## int cgradm ;        /* valeur de l'attr. code du groupe de adm*/

/* ATTENTION : dans le programme principal, il faut assigner les
   variables suivantes, et ceci parce que ce fichier sera inclus
   dans differents modules faisant partie d'un meme programme.
   Une assignation lors de la declaration de la variable cause
   un message d'erreur :

   cadm = 1 ;
   cenv = 2 ;
   cpou = 3 ;
   cgradm = 4 ;

*/
```

```
#  include "stdio.h"
## include "lib_c.qc"
## include "lib_v.qc"


/*****************************************************************
******************************************************************
**          FONCTIONS DE BASE FAISANT APPEL A INGRES            **
******************************************************************
******************************************************************/


/*****************************************************************
******************************************************************
**                    PROCEDURE          OUV_BD                   **
******************************************************************
******************************************************************/

/* Specification : la procedure ouv_bd () procede a l'ouverture
   de la base de donnees 'er' ainsi qu'a la declaration d'un
   nombre de synomymes qui pourront etre utilises dans les
   autres fonctions faisant appel a Ingres. Par exemple 'a'
   pour 'armoire'.
*/


ouv_bd ()

{
/* DEBUT DE LA PROCEDURE */

/* ouverture de la base de donnees */
## ingres er

/* declaration des abreviations des types d'objets en Ingres */
## range of u is utilisateur
## range of gu is gr_util
## range of e is env_rang
## range of a is armoire
## range of t is tiroir
## range of f is farde
## range of p is poubelle
## range of b is boite
## range of gi is gr_info
## range of i is information

/* Remarque : il ne peut y avoir que 10 range declares
   a un moment donne, la declaration d'un onzieme annule
   automatiquement le premier (rotation).
   Le range c (compteur) ne sera declare ici.
   (## range of c is compteur)
*/

   return (0) ;

}  /* FIN DE LA PROCEDURE OUV_BD () */
```

```
/*****************************************************************
*****************************************************************
**                    PROCEDURE          FERM_BD                 **
*****************************************************************
*****************************************************************/

ferm_bd ()

/* Specification : la procedure ferm_bd () procede a la fermeture
   de la base de donnees 'er'.
*/


{
/* DEBUT DE LA PROCEDURE */

/* fermeture de la base de donnees */

## exit
   return (0) ;
}  /* FIN DE LA PROCEDURE FERM_BD () */




/*****************************************************************
*****************************************************************
**                    PROCEDURE      GEN_COMPT (nb)               **
*****************************************************************
*****************************************************************/

/* Specification : gen_compt va lire un nombre dans la table
   compteur, l'incremente de 1 et ecrit cette nouvelle valeur
   dans la table compteur avant de la renvoyer.
*/


gen_compt (nb, code_err)

/* DECLARATION DES PARAMETRES DE LA PROCEDURE */

   long *nb ;                    /* resultat */
   int  *code_err ;              /* resultat */
{
/* DECLARATION DE VARIABLES LOCALES A LA PROCEDURE */
## long temp1 ;                 /* variable temporaire pour le code
                                   pour la declaration Ingres */

/* DEBUT DE LA PROCEDURE */

/* Lecture de la valeur courante de compteur */
## retrieve (temp1 = compteur.code)
```

```
## {
## }

/* il n'y aura normalement jamais qu'un nombre dans la table
   compteur ; ce nombre correspond au code du dernier objet
   qui a ete genere dans l'E.R.
*/

   *nb = ++temp1 ;

## inquire_equel (row_count = "rowcount")

   switch (row_count)
   {
      case 0  : *code_err = GEN_CODE_R ;
                break ;                          /* si pas de code */
      case 1  : *code_err = OK ; break ;
      default : *code_err = GEN_CODE_P ; /* si plusieurs codes */
   }  /* sortie du switch row_count */

   if (*code_err != OK)
   return (0) ;  /* sortie prematuree de gen_compt apres erreur */

/* Ecriture de la nouvelle valeur du compteur */
## replace compteur (code = compteur.code+1)

## inquire_equel (row_count = "rowcount")

   switch (row_count)
   {
      case 0  : *code_err = GEN_CODE_W ;
                break ;              /* si pas de remplacement */
      case 1  : *code_err = OK ; break ;
      default : *code_err = GEN_CODE_P ;
                                  /* si plusieurs remplacements */
   }   /* sortie du switch row_count */

   return (0) ;
}  /* FIN DE LA PROCEDURE GEN_COMPT () */



/*****************************************************************
******************************************************************
**                        PROCEDURE    EXIST                     **
******************************************************************
******************************************************************/

/* Specification : la procedure exist () procede a la verification
   proprement dite de l'existence d'un objet dans l'E.R. sur base
   de son type et de son code. Cette operation se fait au niveau
   physique c'est-a-dire dans l'SGBD cible utilise.
   Dans ce cas il s'agit de l'SGBD INGRES.
*/
```

```
exist (otype, ocode, code_err)

/* DECLARATION DES VARIABLES PARAMETRES */

   int  otype ;              /* donnee */
   long ocode ;              /* donnee */
   int  *code_err ;          /* resultat */

{
/* DECLARATION DES VARIABLES LOCALES A LA PROCEDURE */
##  long temp2 ;    /* variable de travail Ingres : temp(2) parce
                       qu'il existe deja temp1 dans lib_const.qc */

/* DEBUT DE LA PROCEDURE */

/* affectation de le variable ocode a temp2 pour pouvoir
   y affecter une valeur dans la requete Ingres */

   temp2 = ocode ;

/* Traitement du type d'objet particulier */

   switch (otype)

   {
     /*************************************************
     *                EXIST    UTILISATEUR             *
     *************************************************/

   case sutil :

## retrieve (temp2=u.code)
## where u.code = temp2

## inquire_equel (row_count = "rowcount")

   switch (row_count)
   {
      case 0  : *code_err = BD_0_INEXIST ; break ;
      case 1  : *code_err = OK ; break ;
      default : *code_err = BD_CONT_INT ;
   } ;          /* sortie du case row_count */

   break ;      /* sortie du case sutil */


     /*************************************************
     *                EXIST         GR_UTIL            *
     *************************************************/

   case sgr_util :

## retrieve (temp2=gu.code)
## where gu.code = temp2

## inquire_equel (row_count = "rowcount")
```

```
   switch (row_count)
   {
      case 0  : *code_err = BD_0_INEXIST ; break ;
      case 1  : *code_err = OK ; break ;
      default : *code_err = BD_CONT_INT ;
   } ;          /* sortie du case row_count */

   break ;      /* sortie du case sg_util */


     /*************************************************
     *                EXIST      ENV_RANG              *
     *************************************************/

   case senv_rang :

## retrieve (temp2=e.code)
## where e.code = temp2

## inquire_equel (row_count = "rowcount")

   switch (row_count)
   {
      case 0  : *code_err = BD_0_INEXIST ; break ;
      case 1  : *code_err = OK ; break ;
      default : *code_err = BD_CONT_INT ;
   } ;          /* sortie du case row_count */

   break ;      /* sortie du case senv_rang */


     /*************************************************
     *                EXIST      POUBELLE              *
     *************************************************/

   case spou :

## retrieve (temp2=p.code)
## where p.code=temp2

## inquire_equel (row_count = "rowcount")

   switch (row_count)
   {
      case 0  : *code_err = BD_0_INEXIST ; break ;
      case 1  : *code_err = OK ; break ;
      default : *code_err = BD_CONT_INT ;
   } ;          /* sortie du case row_count */

   break ;      /* sortie du case spou */


     /*************************************************
     *                EXIST      BOITE                 *
     *************************************************/
```

```
   case sboi :

## retrieve (temp2=b.code)
## where b.code = temp2

## inquire_equel (row_count = "rowcount")

   switch (row_count)
   {
      case 0  : *code_err = BD_0_INEXIST ; break ;
      case 1  : *code_err = OK ; break ;
      default : *code_err = BD_CONT_INT ;
   } ;          /* sortie du case row_count */

   break ;      /* sortie du case sboi */


     /************************************************
     *                  EXIST      ARMOIRE             *
     ************************************************/

   case sarm :

## retrieve (temp2=a.code)
## where a.code = temp2

## inquire_equel (row_count = "rowcount")

   switch (row_count)
   {
      case 0  : *code_err = BD_0_INEXIST ; break ;
      case 1  : *code_err = OK ; break ;
      default : *code_err = BD_CONT_INT ;
   } ;          /* sortie du case row_count */

   break ;           /* sortie du case sarm */


     /************************************************
     *                  EXIST      TIROIR              *
     ************************************************/

   case stir :

## retrieve (temp2=t.code)
## where t.code = temp2

## inquire_equel (row_count = "rowcount")

   switch (row_count)
   {
      case 0  : *code_err = BD_0_INEXIST ; break ;
      case 1  : *code_err = OK ; break ;
      default : *code_err = BD_CONT_INT ;
   } ;          /* sortie du case row_count */

   break ;      /* sortie du case stir */
```

```
    /*************************************************
    *                  EXIST     FARDE                *
    *************************************************/

   case sfar :

## retrieve (temp2=f.code)
## where f.code = temp2

## inquire_equel (row_count = "rowcount")

   switch (row_count)
   {
      case 0  : *code_err = BD_0_INEXIST ; break ;
      case 1  : *code_err = OK ; break ;
      default : *code_err = BD_CONT_INT ;
   } ;          /* sortie du case row_count */

   break;       /* sortie du case sfar */


    /*************************************************
    *                  EXIST     GR_INFO              *
    *************************************************/

   case sgr_info :

## retrieve (temp2=gi.code)
## where gi.code = temp2

## inquire_equel (row_count = "rowcount")

   switch (row_count)
   {
      case 0  : *code_err = BD_0_INEXIST ; break ;
      case 1  : *code_err = OK ; break ;
      default : *code_err = BD_CONT_INT ;
   } ;          /* sortie du case row_count */

   break ;      /* sortie du case sgi_inf */


    /*************************************************
    *                  EXIST     INFORMATION          *
    *************************************************/

   case sinfo :

## retrieve (temp2=i.code)
## where i.code = temp2

## inquire_equel (row_count = "rowcount")

   switch (row_count)
   {
```

```
      case 0  : *code_err = BD_0_INEXIST ; break ;
      case 1  : *code_err = OK ; break ;
      default : *code_err = BD_CONT_INT ;
   }            /* sortie du case row_count */

   break ;      /* sortie du case sinfo */


     /*************************************************
     *               EXIST     INFORMATION             *
     *************************************************/

   default  :
   *code_err= TCONT_INDEF ;  /* type d'objet non defini */

   }            /* sortie du switch obj->type_object */

   return (0) ;

} /* FIN DE LA PROCEDURE EXIST */



/*****************************************************************
******************************************************************
**                      PROCEDURE    ACCES                      **
******************************************************************
******************************************************************/


/* Specification : la procedure acces () procede a la verification
   proprement dite de l'acces a un objet dans l'E.R. par un
   utilisateur. Cette operation se fait au niveau physique
   c'est-a-dire dans l'SGBD cible utilise.
   Dans ce cas il s'agit de l'SGBD INGRES.
*/


acces (otype, ocode, uposs, code_err)

/* DECLARATION DES VARIABLES PARAMETRES */

   int  otype ;             /* donnee */
   long ocode ;             /* donnee */
   char uposs[LGMAXPOS] ;   /* donnee */
   int  *code_err ;         /* resultat */

{
/* DECLARATION DES VARIABLES LOCALES A LA PROCEDURE */
##  long temp3 ;    /* variable de travail Ingres : temp(2) parce
                       qu'il existe deja temp1 dans lib_const.qc */

/* DEBUT DE LA PROCEDURE */

/* affectation de le variable ocode a temp3 pour pouvoir
```

```
   y affecter une valeur dans la requete Ingres */

   temp3 = ocode ;

/* copie et ajoute du caractere '*' pour la requete retrieve */
   strcpy (cuposs, uposs) ;
   strcat (cuposs, "*") ;


/* Traitement du type d'objet particulier */

   switch (otype)

   {
     /*************************************************
     *                ACCES     ENV_RANG                *
     *************************************************/

     case senv_rang :

     /* Tous les utilisateurs peuvent acceder a l'E.R */
     *code_err = OK ;

     break ;      /* sortie du case senv_rang */


     /*************************************************
     *                 ACCES     POUBELLE              *
     *************************************************/

     case spou :


     /* Tous les utilisateurs peuvent acceder a l'E.R */
     *code_err = OK ;

     break ;      /* sortie du case spou */


     /*************************************************
     *                ACCES     BOITE                  *
     *************************************************/

   case sboi :

## retrieve (temp3=b.code)
## where b.code = temp3
##   and b.poss = cuposs

## inquire_equel (row_count = "rowcount")

   switch (row_count)
   {
      case 0  : *code_err = AC ; break ;
      case 1  : *code_err = OK ; break ;
      default : *code_err = BD_CONT_INT ;
   } ;          /* sortie du case row_count */
```

```
   break ;      /* sortie du case sboi */


     /*************************************************
     *                  ACCES     ARMOIRE              *
     *************************************************/

   case sarm :

## retrieve (temp3=a.code)
## where a.code = temp3
##   and a.poss = cuposs

## inquire_equel (row_count = "rowcount")

   switch (row_count)
   {
      case 0  : *code_err = AC ; break ;
      case 1  : *code_err = OK ; break ;
      default : *code_err = BD_CONT_INT ;
   } ;          /* sortie du case row_count */

   break ;           /* sortie du case sarm */


     /*************************************************
     *                  ACCES     TIROIR               *
     *************************************************/

   case stir :

## retrieve (temp3=t.code)
## where t.code = temp3
##   and t.poss = cuposs

## inquire_equel (row_count = "rowcount")

   switch (row_count)
   {
      case 0  : *code_err = AC ; break ;
      case 1  : *code_err = OK ; break ;
      default : *code_err = BD_CONT_INT ;
   } ;          /* sortie du case row_count */

   break ;      /* sortie du case stir */


     /*************************************************
     *                  ACCES     FARDE                *
     *************************************************/

   case sfar :

## retrieve (temp3=f.code)
## where f.code = temp3
##   and f.poss = cuposs
```

```
## inquire_equel (row_count = "rowcount")

   switch (row_count)
   {
      case 0  : *code_err = AC ; break ;
      case 1  : *code_err = OK ; break ;
      default : *code_err = BD_CONT_INT ;
   } ;          /* sortie du case row_count */

   break;       /* sortie du case sfar */


     /*************************************************
     *                  ACCES    INDEFINI               *
     *************************************************/

   default  :
   *code_err= TOB_AC ;  /* type d'objet non defini */

   }            /* sortie du switch otype */

   return (0) ;

} /* FIN DE LA PROCEDURE ACCES */



/*****************************************************************
******************************************************************
**                    PROCEDURE    ACCES_CON                     **
******************************************************************
******************************************************************/

/* Specification : la procedure acces () procede a la verification
   proprement dite de l'acces en consultation a une information
   ou un groupe d'informations de l'E.R. par un utilisateur.
   Cette operation se fait au niveau physique c'est-a-dire dans
   l'SGBD cible utilise. Dans ce cas il s'agit de l'SGBD INGRES.
*/


acces_con (otype, ocode, uposs, code_err)

/* DECLARATION DES VARIABLES PARAMETRES */

   int  otype ;             /* donnee */
   long ocode ;             /* donnee */
   char uposs[LGMAXPOS] ;   /* donnee */
   int  *code_err ;         /* resultat */

{
/* DECLARATION DES VARIABLES LOCALES A LA PROCEDURE */
##  long temp4 ;    /* variable de travail Ingres : temp(2) parce
                       qu'il existe deja temp1 dans lib_const.qc */
```

```
/* DEBUT DE LA PROCEDURE */

/* affectation de le variable ocode a temp4 pour pouvoir
   y affecter une valeur dans la requete Ingres */

   temp4 = ocode ;

/* copie et ajoute du caractere '*' pour la requete retrieve */
   strcpy (cuposs, uposs) ;
   strcat (cuposs, "*") ;


/* Traitement du type d'objet particulier */

   switch (otype)

   {
     /*************************************************
     *                 ACCES_CON    GR_INFO            *
     *************************************************/

   case sgr_info :

## retrieve (temp4=gi.code)
## where gi.code = temp4
##   and gi.poss_con = cuposs

## inquire_equel (row_count = "rowcount")

   switch (row_count)
   {
      case 0  : *code_err = AC_CONSUL ; break ;
      case 1  : *code_err = OK ; break ;
      default : *code_err = BD_CONT_INT ;
   } ;          /* sortie du case row_count */

   break ;      /* sortie du case sgi_inf */


     /*************************************************
     *                ACCES_CON     INFORMATION        *
     *************************************************/

   case sinfo :

## retrieve (temp4=i.code)
## where i.code = temp4
##   and i.poss_con = cuposs

## inquire_equel (row_count = "rowcount")

   switch (row_count)
   {
      case 0  : *code_err = AC_CONSUL ; break ;
      case 1  : *code_err = OK ; break ;
      default : *code_err = BD_CONT_INT ;
```

```
    }           /* sortie du case row_count */

    break ;     /* sortie du case sinfo */

      /*************************************************
      *                ACCES_CON    INDEFINI            *
      *************************************************/

    default  :
    *code_err= TCONT_INDEF ;  /* type d'objet non defini */

    }           /* sortie du switch otype */

    return (0) ;

} /* FIN DE LA PROCEDURE ACCES_CON */



/*****************************************************************
******************************************************************
**                    PROCEDURE     ACCES_MAJ                    **
******************************************************************
******************************************************************/

/* Specification : la procedure acces () procede a la verification
   proprement dite de l'acces en mise a jour a une information
   ou a un groupe d'informations de l'E.R. par un utilisateur.
   Cette operation se fait au niveau physique c'est-a-dire dans
   l'SGBD cible utilise. Dans ce cas il s'agit de l'SGBD INGRES.
*/


acces_maj (otype, ocode, uposs, code_err)

/* DECLARATION DES VARIABLES PARAMETRES */

    int  otype ;            /* donnee */
    long ocode ;            /* donnee */
    char uposs[LGMAXPOS] ; /* donnee */
    int  *code_err ;        /* resultat */

{
/* DECLARATION DES VARIABLES LOCALES A LA PROCEDURE */
##  long temp5 ;    /* variable de travail Ingres : temp(2) parce
                       qu'il existe deja temp1 dans lib_const.qc */

/* DEBUT DE LA PROCEDURE */

/* affectation de le variable ocode a temp5 pour pouvoir
   y affecter une valeur dans la requete Ingres */

    temp5 = ocode ;

/* copie et ajoute du caractere '*' pour la requete retrieve */
```

```
   strcpy (cuposs, uposs) ;
   strcat (cuposs, "*") ;


/* Traitement du type d'objet particulier */

   switch (otype)

   {
     /*************************************************
     *                ACCES_MAJ    GR_INFO             *
     *************************************************/

   case sgr_info :

## retrieve (temp5=gi.code)
## where gi.code = temp5
##   and gi.poss_maj = cuposs

## inquire_equel (row_count = "rowcount")

   switch (row_count)
   {
      case 0  : *code_err = AC_MAJ ; break ;
      case 1  : *code_err = OK ; break ;
      default : *code_err = BD_CONT_INT ;
   } ;          /* sortie du case row_count */

   break ;      /* sortie du case sgi_inf */


     /*************************************************
     *                ACCES_MAJ     INFORMATION        *
     *************************************************/

   case sinfo :

## retrieve (temp5=i.code)
## where i.code = temp5
##   and i.poss_maj = cuposs

## inquire_equel (row_count = "rowcount")

   switch (row_count)
   {
      case 0  : *code_err = AC_MAJ ; break ;
      case 1  : *code_err = OK ; break ;
      default : *code_err = BD_CONT_INT ;
   }            /* sortie du case row_count */

   break ;      /* sortie du case sinfo */

     /*************************************************
     *                ACCES_MAJ    INDEFINI            *
     *************************************************/

   default  :
```

```
   *code_err= TCONT_INDEF ;  /* type d'objet non defini */

   }            /* sortie du switch otype */

   return (0) ;

} /* FIN DE LA PROCEDURE ACCES_MAJ */



/*****************************************************************
*****************************************************************
**                    PROCEDURE       PROPR                    **
*****************************************************************
*****************************************************************/

/* Specification : la procedure propr () donne les proprietes
   d'un objet dans l'E.R. Cette operation se fait au niveau
   physique c'est-a-dire dans l'SGBD cible utilise.
   Dans ce cas il s'agit de l'SGBD INGRES.
*/


propr (obj, code_err)

/* DECLARATION DES VARIABLES PARAMETRES */

   tobject  *obj  ;        /* donnee et resultat */
   int  *code_err ;        /* resultat */

{
/* DECLARATION DES VARIABLES LOCALES A LA PROCEDURE */
##  struct ituti     *uobj ;
##  struct itgruti   *guobj ;
##  struct itenvran  *eobj ;
##  struct itarm     *aobj ;
##  struct ittir     *tobj ;
##  struct itfar     *fobj ;
##  struct itpou     *pobj ;
##  struct itboi     *bobj ;
##  struct itgrinf   *giobj ;
##  struct itinf     *iobj ;

/* DEBUT DE LA PROCEDURE */

/* affectation d'une partie de l'objet a une variable
   particuliere pour faire l'interface avec Ingres.
   Ingres n'accepte que des structures d'un niveau et
   tous les noms des champs des structures doivent etre
   differents. */

   switch (obj->type_object)
   {
   case sutil      : uobj = &(obj->o.u) ;
                     break ;
```

```
   case sgr_util   : guobj = &(obj->o.gu) ;
                     break ;
   case senv_rang : eobj = &(obj->o.e) ;
                     break ;
   case sarm       : aobj = &(obj->o.a) ;
                     break ;
   case stir       : tobj = &(obj->o.t) ;
                     break ;
   case sfar       : fobj = &(obj->o.f) ;
                     break ;
   case spou       : pobj = &(obj->o.p) ;
                     break ;
   case sboi       : bobj = &(obj->o.b) ;
                     break ;
   case sgr_info   : giobj = &(obj->o.gi) ;
                     break ;
   case sinfo      : iobj = &(obj->o.i) ;
                     break ;
   default         : break ;
   }    /* sortie du switch obj->type_object */

/* Traitement du type d'objet particulier */

   switch (obj->type_object)

   {
     /*************************************************
     *               PROPR      UTILISATEUR             *
     *************************************************/

   case sutil :

## retrieve (uobj->unom=u.nom,
##           uobj->upwd=u.pwd,uobj->ucrdate=u.crdate,
##           uobj->ucreat=u.creat,uobj->uperm=u.perm,
##           uobj->ugroupe=u.groupe)
## where u.code = uobj->ucode

## inquire_equel (row_count = "rowcount")

   switch (row_count)
   {
      case 0  : *code_err = BD_0_INEXIST ; break ;
      case 1  : *code_err = OK ; break ;
      default : *code_err = BD_CONT_INT ;
   } ;          /* sortie du case row_count */

   break ;      /* sortie du case sutil */


     /*************************************************
     *               PROPR          GR_UTIL             *
     *************************************************/

   case sgr_util :

## retrieve (guobj->gunom=gu.nom,guobj->gucrdate=gu.crdate,
```

```
##            guobj->gucreat=gu.creat)
## where gu.code = guobj->gucode

## inquire_equel (row_count = "rowcount")

   switch (row_count)
   {
      case 0  : *code_err = BD_0_INEXIST ; break ;
      case 1  : *code_err = OK ; break ;
      default : *code_err = BD_CONT_INT ;
   } ;          /* sortie du case row_count */

   break ;      /* sortie du case sg_util */


     /*************************************************
     *                PROPR        ENV_RANG            *
     *************************************************/

   case senv_rang :

## retrieve (eobj->enom=e.nom,
##           eobj->ecrdate=e.crdate,eobj->ecreat=e.creat,
##           eobj->edesc=e.desc,eobj->estconf=e.stconf,
##           eobj->eposs=e.poss)
## where e.code = eobj->ecode

## inquire_equel (row_count = "rowcount")

   switch (row_count)
   {
      case 0  : *code_err = BD_0_INEXIST ; break ;
      case 1  : *code_err = OK ; break ;
      default : *code_err = BD_CONT_INT ;
   } ;          /* sortie du case row_count */

   break ;      /* sortie du case senv_rang */


     /*************************************************
     *                PROPR        POUBELLE            *
     *************************************************/

   case spou :

## retrieve (pobj->pnom=p.nom,
##           pobj->pcrdate=p.crdate,pobj->pcreat=p.creat,
##           pobj->pdesc=p.desc,pobj->pstconf=p.stconf,
##           pobj->pposs=p.poss)
## where p.code=pobj->pcode

## inquire_equel (row_count = "rowcount")

   switch (row_count)
   {
      case 0  : *code_err = BD_0_INEXIST ; break ;
      case 1  : *code_err = OK ; break ;
```

```
      default : *code_err = BD_CONT_INT ;
   } ;          /* sortie du case row_count */

   break ;      /* sortie du case spou */


     /*************************************************
     *                    PROPR     BOITE              *
     *************************************************/

   case sboi :

## retrieve (bobj->bnom=b.nom,
##           bobj->bcrdate=b.crdate,bobj->bcreat=b.creat,
##           bobj->bdesc=b.desc,bobj->bstconf=b.stconf,
##           bobj->btype=b.type,bobj->bposs=b.poss)
## where b.code = bobj->bcode

## inquire_equel (row_count = "rowcount")

   switch (row_count)
   {
      case 0  : *code_err = BD_0_INEXIST ; break ;
      case 1  : *code_err = OK ; break ;
      default : *code_err = BD_CONT_INT ;
   } ;          /* sortie du case row_count */

   break ;      /* sortie du case sboi */


     /*************************************************
     *                    PROPR     ARMOIRE            *
     *************************************************/

   case sarm :

## retrieve (aobj->anom=a.nom,
##           aobj->acrdate=a.crdate,aobj->acreat=a.creat,
##           aobj->adesc=a.desc,aobj->astconf=a.stconf,
##           aobj->atype=a.type,aobj->aposs=a.poss)
## where a.code = aobj->acode

## inquire_equel (row_count = "rowcount")

   switch (row_count)
   {
      case 0  : *code_err = BD_0_INEXIST ; break ;
      case 1  : *code_err = OK ; break ;
      default : *code_err = BD_CONT_INT ;
   } ;          /* sortie du case row_count */

   break ;          /* sortie du case sarm */


     /*************************************************
     *                    PROPR     TIROIR             *
     *************************************************/
```

```
   case stir :

## retrieve (tobj->tnom=t.nom,
##           tobj->tcrdate=t.crdate,tobj->tcreat=t.creat,
##           tobj->tdesc=t.desc,tobj->tstconf=t.stconf,
##           tobj->ttype=t.type,tobj->tposs=t.poss,
##           tobj->tcont_c=t.code_armoire)
## where t.code = tobj->tcode

## inquire_equel (row_count = "rowcount")

   switch (row_count)
   {
      case 0  : *code_err = BD_0_INEXIST ; break ;
      case 1  : *code_err = OK ; break ;
      default : *code_err = BD_CONT_INT ;
   } ;          /* sortie du case row_count */

   break ;      /* sortie du case stir */


     /*************************************************
     *                  PROPR       FARDE              *
     *************************************************/

   case sfar :

## retrieve (fobj->fnom=f.nom,
##           fobj->fcrdate=f.crdate,fobj->fcreat=f.creat,
##           fobj->fdesc=f.desc,fobj->fstconf=f.stconf,
##           fobj->ftype=f.type,fobj->fposs=f.poss,
##           fobj->fcont_c=f.code_tiroir)
## where f.code = fobj->fcode

## inquire_equel (row_count = "rowcount")

   switch (row_count)
   {
      case 0  : *code_err = BD_0_INEXIST ; break ;
      case 1  : *code_err = OK ; break ;
      default : *code_err = BD_CONT_INT ;
   } ;          /* sortie du case row_count */

   break;       /* sortie du case sfar */


     /*************************************************
     *                   PROPR     GR_INFO             *
     *************************************************/

   case sgr_info :

## retrieve (giobj->ginom=gi.nom,
##           giobj->gicrdate=gi.crdate,giobj->gicreat=gi.creat,
##           giobj->gipresence=gi.presence,
##           giobj->gidesc=gi.desc,giobj->gistconf=gi.stconf,
```

```
##            giobj->gitype=gi.type,
##            giobj->giposs_con=gi.poss_con,
##            giobj->giposs_maj=gi.poss_maj,
##            giobj->gicont_t=gi.cont_t,
##            giobj->gicont_c=gi.cont_c)
## where gi.code = giobj->gicode

## inquire_equel (row_count = "rowcount")

   switch (row_count)
   {
      case 0  : *code_err = BD_0_INEXIST ; break ;
      case 1  : *code_err = OK ; break ;
      default : *code_err = BD_CONT_INT ;
   } ;          /* sortie du case row_count */

   break ;      /* sortie du case sgi_inf */


     /*************************************************
     *                PROPR     INFORMATION            *
     *************************************************/

   case sinfo :

## retrieve (iobj->inom=i.nom,
##            iobj->inature=i.nature,iobj->ipresence=i.presence,
##            iobj->icrdate=i.crdate,iobj->icreat=i.creat,
##            iobj->icodate=i.codate,iobj->iconsult=i.consult,
##            iobj->imodate=i.modate,iobj->imodificat=i.modificat,
##            iobj->idesc=i.desc,iobj->istconf=i.stconf,
##            iobj->itype=i.type,iobj->iposs_con=i.poss_con,
##            iobj->iposs_maj=i.poss_maj,iobj->iexped=i.exped,
##            iobj->imots_cles=i.mots_cles,
##            iobj->iref_stck=i.ref_stck,
##            iobj->iref_orig=i.ref_orig,iobj->icont_t=i.cont_t,
##            iobj->icont_c=i.cont_c)
## where i.code = iobj->icode

## inquire_equel (row_count = "rowcount")

   switch (row_count)
   {
      case 0  : *code_err = BD_0_INEXIST ; break ;
      case 1  : *code_err = OK ; break ;
      default : *code_err = BD_CONT_INT ;
   }            /* sortie du case row_count */

   break ;      /* sortie du case sinfo */

   }            /* sortie du switch obj->type_object */

} /* FIN DE LA PROCEDURE PROPR */
```

```
# include "stdio.h"
# include "lib_c.c"


/*****************************************************************
 *****************************************************************
 **                      PROCEDURE      INITU                     **
 *****************************************************************
 *****************************************************************/

/* Specification : initiatlisation d'un objet de type
   utilisateur. */

initu (uobj)

   struct ituti *uobj ;

{
/* DEBUT DE LA PROCEDURE */

   uobj->unom[0] = '\0' ;
   uobj->ucode = 0 ;
   uobj->upwd[0]  = '\0' ;
   uobj->ucrdate[0] = '\0' ;
   uobj->ucreat = 0 ;
   uobj->uperm[0] = '\0' ;
   uobj->ugroupe[0] = '\0' ;
}
/* FIN DE LA PROCEDURE INITU () */


/*****************************************************************
 *****************************************************************
 **                      PROCEDURE      INITGU                    **
 *****************************************************************
 *****************************************************************/

/* Specification : initiatlisation d'un objet de type
   groupe_utilisateur. */

initgu (guobj)

   struct itgruti  *guobj ;

{
/* DEBUT DE LA PROCEDURE */

   guobj->gunom[0] = '\0' ;
   guobj->gucode = 0 ;
   guobj->gucrdate[0] = '\0' ;
   guobj->gucreat = 0 ;
}
/* FIN DE LA PROCEDURE INITGU () */
```

```
/*****************************************************************
 *****************************************************************
 **                        PROCEDURE      INITE                   **
 *****************************************************************
 *****************************************************************/

/* Specification : initiatlisation d'un objet de type env_rang. */

inite (eobj)

   struct itenvran *eobj ;

{
/* DEBUT DE LA PROCEDURE */

   eobj->enom[0] = '\0' ;
   eobj->ecode = 0 ;
   eobj->ecrdate[0] = '\0' ;
   eobj->ecreat = 0 ;
   eobj->edesc[0] = '\0' ;
   eobj->estconf[0] = '\0' ;
   eobj->eposs[0] = '\0' ;
}
/* FIN DE LA PROCEDURE INITE () */



/*****************************************************************
 *****************************************************************
 **                        PROCEDURE      INITA                   **
 *****************************************************************
 *****************************************************************/

/* Specification : initiatlisation d'un objet de type armoire. */

inita (aobj)

   struct itarm    *aobj ;

{
/* DEBUT DE LA PROCEDURE */

   aobj->anom[0] = '\0' ;
   aobj->acode = 0 ;
   aobj->acrdate[0] = '\0' ;
   aobj->acreat = 0 ;
   aobj->adesc[0] = '\0' ;
   aobj->astconf[0] = '\0' ;
   aobj->atype[0] = '\0' ;
   aobj->aposs[0] = '\0' ;
}
/* FIN DE LA PROCEDURE INITA () */
```

```
/*****************************************************************
 *****************************************************************
 **                      PROCEDURE      INITT                     **
 *****************************************************************
 *****************************************************************/

/* Specification : initiatlisation d'un objet de type tiroir. */

initt (tobj)

   struct ittir     *tobj ;

{
/* DEBUT DE LA PROCEDURE */

   tobj->tnom[0] = '\0' ;
   tobj->tcode = 0 ;
   tobj->tcrdate[0] = '\0' ;
   tobj->tcreat = 0 ;
   tobj->tdesc[0] = '\0' ;
   tobj->tstconf[0] = '\0' ;
   tobj->ttype[0] = '\0' ;
   tobj->tposs[0] = '\0' ;
   tobj->tcont_c = 0 ;
}
/* FIN DE LA PROCEDURE INITT () */


/*****************************************************************
 *****************************************************************
 **                      PROCEDURE      INITF                     **
 *****************************************************************
 *****************************************************************/

/* Specification : initiatlisation d'un objet de type farde. */

initf (fobj)

   struct itfar     *fobj ;

{
/* DEBUT DE LA PROCEDURE */

   fobj->fnom[0] = '\0' ;
   fobj->fcode = 0 ;
   fobj->fcrdate[0] = '\0' ;
   fobj->fcreat = 0 ;
   fobj->fdesc[0] = '\0' ;
   fobj->fstconf[0] = '\0' ;
   fobj->ftype[0] = '\0' ;
   fobj->fposs[0] = '\0' ;
   fobj->fcont_c = 0 ;
}
/* FIN DE LA PROCEDURE INITF () */
```

```
/*************************************************************************
 *************************************************************************
 **                        PROCEDURE      INITP                         **
 *************************************************************************
 *************************************************************************/

/* Specification : initiatlisation d'un objet de type poubelle. */

initp (pobj)

   struct itpou     *pobj ;

{
/* DEBUT DE LA PROCEDURE */

   pobj->pnom[0] = '\0' ;
   pobj->pcode = 0 ;
   pobj->pcrdate[0] = '\0' ;
   pobj->pcreat = 0 ;
   pobj->pdesc[0] = '\0' ;
   pobj->pstconf[0] = '\0' ;
   pobj->pposs[0] = '\0' ;
}
/* FIN DE LA PROCEDURE INITP () */



/*************************************************************************
 *************************************************************************
 **                        PROCEDURE      INITB                         **
 *************************************************************************
 *************************************************************************/

/* Specification : initiatlisation d'un objet de type boite. */

initb (bobj)

   struct itboi     *bobj ;

{
/* DEBUT DE LA PROCEDURE */

   bobj->bnom[0] = '\0' ;
   bobj->bcode = 0 ;
   bobj->bcrdate[0] = '\0' ;
   bobj->bcreat = 0 ;
   bobj->bdesc[0] = '\0' ;
   bobj->bstconf[0] = '\0' ;
   bobj->btype[0] = '\0' ;
   bobj->bposs[0] = '\0'
}
/* FIN DE LA PROCEDURE INITB () */
```

```
/*****************************************************************
 *****************************************************************
 **                      PROCEDURE      INITGI                    **
 *****************************************************************
 *****************************************************************/

/* Specification : initiatlisation d'un objet de type
   groupe_information. */

initgi (giobj)

   struct itgrinf  *giobj ;

{
/* DEBUT DE LA PROCEDURE */

   giobj->ginom[0] = '\0' ;
   giobj->gicode = 0 ;
   giobj->gicrdate[0] = '\0' ;
   giobj->gicreat = 0 ;
   giobj->gipresence = 0 ;
   giobj->gidesc[0] = '\0' ;
   giobj->gistconf[0] = '\0' ;
   giobj->gitype[0] = '\0' ;
   giobj->giposs_con[0] = '\0' ;
   giobj->giposs_maj[0] = '\0' ;
   giobj->gicont_t = 0 ;
   giobj->gicont_c = 0 ;
}
/* FIN DE LA PROCEDURE INITGI () */



/*****************************************************************
 *****************************************************************
 **                      PROCEDURE      INITI                     **
 *****************************************************************
 *****************************************************************/

/* Specification : initiatlisation d'un objet de type
   information. */

initi (iobj)

   struct itinf    *iobj ;

{
/* DEBUT DE LA PROCEDURE */

   iobj->inom[0] = '\0' ;
   iobj->icode = 0 ;
   iobj->inature[0] = '\0' ;
   iobj->icrdate[0] = '\0' ;
   iobj->icreat = 0 ;
   iobj->ipresence = 0 ;
```

```
   iobj->icodate[0] = '\0';
   iobj->iconsult = 0 ;
   iobj->imodate[0] = '\0' ;
   iobj->imodificat = 0 ;
   iobj->idesc[0] = '\0' ;
   iobj->istconf[0] = '\0' ;
   iobj->itype[0] = '\0' ;
   iobj->iposs_con[0] = '\0' ;
   iobj->iposs_maj[0] = '\0' ;
   iobj->iexped[0] = '\0' ;
   iobj->imots_cles[0] = '\0' ;
   iobj->iref_stck[0] = '\0' ;
   iobj->iref_orig[0] = '\0' ;
   iobj->icont_t = 0 ;
   iobj->icont_c = 0 ;
}
/* FIN DE LA PROCEDURE INITI () */



/*****************************************************************
 *****************************************************************
 **              PROCEDURE     STR_IN                           **
 *****************************************************************
 *****************************************************************/

/* Specification : cette procedure verifie si la chaine de
   caracteres str1 se trouve dans str2. Le resultat est renvoye
   par la fonction :
   Si le str1 se trouve dans str2 la valeur est 1 (true)
   Sinon la valeur est 0 (false)
*/

str_in (str1, str2)

/* DECLARATION DES PARAMETRES DE LA PROCEDURE */

   char str1[] ;          /* donnee */
   char str2[] ;          /* donnee */

{
/* DECLARATION DES VARIABLES LOCALES A LA PROCEDURE */

   int len1, len2 ;       /* longueur des chaines str1 et str2 */
   int ind1, ind2 ;       /* positions temporaires */
   int temp ;             /* variable temporaire de recherche */
   int espoir = 0 ;       /* indicateur d'espoir a faux */
   int trouve = 0 ;       /* indicateur de fin a faux */

/* DEBUT DE LA PROCEDURE */

   len1 = strlen (str1) ;
   len2 = strlen (str2) ;

   if (len1 > len2)
```

```
    return (0) ;        /* faux */

    ind1 = 0 ; ind2 = 0 ;

    while ( (ind2 < len2) && ! trouve)
    { if ( str1[ind1] == str2[ind2] )
      {  espoir = 1 ;
         temp = ++ind2 ;
         ++ind1 ;
         while ( (ind1 < len1) && (temp < len2 ) && espoir )
         {  if (str1[ind1] != str2[temp])
               espoir = 0 ;
            ind1++ ;
            temp++ ;
         }
         if (ind1 == len1)
            trouve = espoir ;
         ind1 = 0 ;
      }
    ind2++ ;
    }
    return (trouve) ;

}
/* FIN DE LA PROCEDURE STR_IN */
```

```
# include "stdio.h"
# include "lib_c.c"


/* Specification : ce fichier lib_t.c contient un certain
   nombres de procedures qui permettront de tester les
   differents modules de l'application en affichant les
   proprietes de objets crees ou affectes. */


/*****************************************************************
*****************************************************************
**                  PROCEDURE         PR_UTI                     **
*****************************************************************
*****************************************************************/

/* Specification : cette procedure permet d'afficher les
   proprietes d'un objet de type utilisateur sur la sortie
   standard (ecran).
*/

pr_uti (uobj,nb)

/* DECLARATION DES VARIABLES PARAMETRES */

   struct ituti uobj ;          /* donnee */
   int    nb ;                  /* donnee */

{
/* DEBUT DE LA PROCEDURE PR_UTI */

   if (nb != 0 )  printf ("%d. \n", nb ) ;
   printf ("    nom     : %s \n", uobj.unom) ;
   printf ("    code    : %d \n", uobj.ucode) ;
   printf ("    crdate  : %s \n", uobj.ucrdate) ;
   printf ("    creat   : %d \n", uobj.ucreat) ;
   printf ("    perm    : %s \n", uobj.uperm) ;
   printf ("    groupe  : %s \n\n", uobj.ugroupe) ;

   return (0) ;

}  /* FIN DE LA PROCEDURE PR_UTI */


/*****************************************************************
*****************************************************************
**                  PROCEDURE         PR_GR_UTI                  **
*****************************************************************
*****************************************************************/

/* Specification : cette procedure permet d'afficher les
   proprietes d'un objet de type groupe_utilisateur sur la
   sortie standard (ecran).
*/
```

```
pr_gr_uti (guobj,nb)

/* DECLARATION DES VARIABLES PARAMETRES */

   struct itgruti guobj ;        /* donnee */
   int    nb ;                   /* donnee */

{
/* DEBUT DE LA PROCEDURE PR_GR_UTI */

   if (nb != 0 )  printf ("%d. \n", nb ) ;
   printf ("   nom     : %s \n", guobj.gunom) ;
   printf ("   code    : %d \n", guobj.gucode) ;
   printf ("   crdate  : %s \n", guobj.gucrdate) ;
   printf ("   creat   : %d \n\n", guobj.gucreat) ;

   return (0) ;

}  /* FIN DE LA PROCEDURE PR_GR_UTI */


/*****************************************************************
******************************************************************
**                  PROCEDURE         PR_ENV_RAN                 **
******************************************************************
******************************************************************/

/* Specification : cette procedure permet d'afficher les
   proprietes d'un objet de type enr_rang sur la sortie
   standard (ecran).
*/

pr_env_ran (eobj,nb)

/* DECLARATION DES VARIABLES PARAMETRES */

   struct itenvran eobj ;        /* donnee */
   int    nb ;                   /* donnee */

{
/* DEBUT DE LA PROCEDURE PR_ENV_RAN */

   if (nb != 0 )  printf ("%d. \n", nb ) ;
   printf ("   nom     : %s \n", eobj.enom) ;
   printf ("   code    : %d \n", eobj.ecode) ;
   printf ("   crdate  : %s \n", eobj.ecrdate) ;
   printf ("   creat   : %d \n", eobj.ecreat) ;
   printf ("   desc    : %s \n", eobj.edesc) ;
   printf ("   stconf  : %s \n", eobj.estconf) ;
   printf ("   poss    : %s \n\n", eobj.eposs) ;

   return (0) ;

}  /* FIN DE LA PROCEDURE PR_ENV_RAN */
```

```
/*****************************************************************
*****************************************************************
**                  PROCEDURE         PR_ARM                     **
*****************************************************************
*****************************************************************/

/* Specification : cette procedure permet d'afficher les
   proprietes d'un objet de type armoire sur la sortie
   standard (ecran).
*/

pr_arm (aobj,nb)

/* DECLARATION DES VARIABLES PARAMETRES */

   struct itarm aobj ;          /* donnee */
   int     nb ;                 /* donnee */

{
/* DEBUT DE LA PROCEDURE PR_ARM */

   if (nb != 0 )  printf ("%d. \n", nb ) ;
   printf ("    nom     : %s \n", aobj.anom) ;
   printf ("    code    : %d \n", aobj.acode) ;
   printf ("    crdate  : %s \n", aobj.acrdate) ;
   printf ("    creat   : %d \n", aobj.acreat) ;
   printf ("    desc    : %s \n", aobj.adesc) ;
   printf ("    stconf  : %s \n", aobj.astconf) ;
   printf ("    type    : %s \n", aobj.atype) ;
   printf ("    poss    : %s \n\n", aobj.aposs) ;

   return (0) ;

}  /* FIN DE LA PROCEDURE PR_ARM */


/*****************************************************************
*****************************************************************
**                  PROCEDURE         PR_TIR                     **
*****************************************************************
*****************************************************************/

/* Specification : cette procedure permet d'afficher les
   proprietes d'un objet de type tiroir sur la sortie
   standard (ecran).
*/

pr_tir (tobj,nb)

/* DECLARATION DES VARIABLES PARAMETRES */

   struct ittir tobj ;          /* donnee */
   int     nb ;                 /* donnee */

{
```

```
/* DEBUT DE LA PROCEDURE PR_TIR */

   if (nb != 0 )  printf ("%d. \n", nb ) ;
   printf ("    nom       : %s \n", tobj.tnom) ;
   printf ("    code      : %d \n", tobj.tcode) ;
   printf ("    crdate    : %s \n", tobj.tcrdate) ;
   printf ("    creat     : %d \n", tobj.tcreat) ;
   printf ("    desc      : %s \n", tobj.tdesc) ;
   printf ("    stconf    : %s \n", tobj.tstconf) ;
   printf ("    type      : %s \n", tobj.ttype) ;
   printf ("    poss      : %s \n", tobj.tposs) ;
   printf ("    code_arm  : %d \n\n", tobj.tcont_c) ;

   return (0) ;

}  /* FIN DE LA PROCEDURE PR_TIR */


/*****************************************************************
******************************************************************
**                  PROCEDURE         PR_FAR                     **
******************************************************************
*****************************************************************/

/* Specification : cette procedure permet d'afficher les
   proprietes d'un objet de type farde sur la sortie
   standard (ecran).
*/

pr_far (fobj,nb)

/* DECLARATION DES VARIABLES PARAMETRES */

   struct itfar fobj ;          /* donnee */
   int    nb ;                  /* donnee */

{
/* DEBUT DE LA PROCEDURE PR_FAR */

   if (nb != 0 )  printf ("%d. \n", nb ) ;
   printf ("    nom       : %s \n", fobj.fnom) ;
   printf ("    code      : %d \n", fobj.fcode) ;
   printf ("    crdate    : %s \n", fobj.fcrdate) ;
   printf ("    creat     : %d \n", fobj.fcreat) ;
   printf ("    desc      : %s \n", fobj.fdesc) ;
   printf ("    stconf    : %s \n", fobj.fstconf) ;
   printf ("    type      : %s \n", fobj.ftype) ;
   printf ("    poss      : %s \n", fobj.fposs) ;
   printf ("    code_tir  : %d \n\n", fobj.fcont_c) ;

   return (0) ;

}  /* FIN DE LA PROCEDURE PR_FAR */
```

```
/*************************************************************
*************************************************************
**                  PROCEDURE        PR_POU                  **
*************************************************************
*************************************************************/

/* Specification : cette procedure permet d'afficher les
   proprietes d'un objet de type poubelle sur la sortie
   standard (ecran).
*/

pr_pou (pobj,nb)

/* DECLARATION DES VARIABLES PARAMETRES */

   struct itpou pobj ;          /* donnee */
   int    nb ;                  /* donnee */

{
/* DEBUT DE LA PROCEDURE PR_POU */

   if (nb != 0 )  printf ("%d. \n", nb ) ;
   printf ("   nom       : %s \n", pobj.pnom) ;
   printf ("   code      : %d \n", pobj.pcode) ;
   printf ("   crdate    : %s \n", pobj.pcrdate) ;
   printf ("   creat     : %d \n", pobj.pcreat) ;
   printf ("   desc      : %s \n", pobj.pdesc) ;
   printf ("   stconf    : %s \n", pobj.pstconf) ;
   printf ("   poss      : %s \n\n", pobj.pposs) ;

   return (0) ;

}  /* FIN DE LA PROCEDURE PR_POU */


/*************************************************************
*************************************************************
**                  PROCEDURE        PR_BOI                  **
*************************************************************
*************************************************************/

/* Specification : cette procedure permet d'afficher les
   proprietes d'un objet de type boite sur la sortie
   standard (ecran).
*/

pr_boi (bobj,nb)

/* DECLARATION DES VARIABLES PARAMETRES */

   struct itboi bobj ;          /* donnee */
   int    nb ;                  /* donnee */

{
/* DEBUT DE LA PROCEDURE PR_BOI */

   if (nb != 0 )  printf ("%d. \n", nb ) ;
```

```
   printf ("    nom       : %s \n", bobj.bnom) ;
   printf ("    code      : %d \n", bobj.bcode) ;
   printf ("    crdate    : %s \n", bobj.bcrdate) ;
   printf ("    creat     : %d \n", bobj.bcreat) ;
   printf ("    desc      : %s \n", bobj.bdesc) ;
   printf ("    stconf    : %s \n", bobj.bstconf) ;
   printf ("    type      : %s \n", bobj.btype) ;
   printf ("    poss      : %s \n\n", bobj.bposs) ;

   return (0) ;

}  /* FIN DE LA PROCEDURE PR_BOI */


/*************************************************************
**************************************************************
**                  PROCEDURE         PR_GR_INF                **
**************************************************************
**************************************************************/

/* Specification : cette procedure permet d'afficher les
   proprietes d'un objet de type groupe_information sur la
   sortie standard (ecran).
*/

pr_gr_inf (giobj,nb)

/* DECLARATION DES VARIABLES PARAMETRES */

   struct itgrinf giobj ;        /* donnee */
   int    nb ;                   /* donnee */

{
/* DEBUT DE LA PROCEDURE PR_GR_INF */

   if (nb != 0 )  printf ("%d. \n", nb ) ;
   printf ("    nom       : %s \n", giobj.ginom) ;
   printf ("    code      : %d \n", giobj.gicode) ;
   printf ("    crdate    : %s \n", giobj.gicrdate) ;
   printf ("    creat     : %d \n", giobj.gicreat) ;
   printf ("    presence  : %d \n", giobj.gipresence) ;
   printf ("    desc      : %s \n", giobj.gidesc) ;
   printf ("    stconf    : %s \n", giobj.gistconf) ;
   printf ("    type      : %s \n", giobj.gitype) ;
   printf ("    poss_con  : %s \n", giobj.giposs_con) ;
   printf ("    poss_maj  : %s \n", giobj.giposs_maj) ;
   printf ("    code_t    : %d \n", giobj.gicont_t) ;
   printf ("    code_c    : %d \n\n", giobj.gicont_c) ;

   return (0) ;

}  /* FIN DE LA PROCEDURE PR_GR_INF */
```

```
/*****************************************************************
*****************************************************************
**                    PROCEDURE          PR_INF                  **
*****************************************************************
*****************************************************************/

/* Specification : cette procedure permet d'afficher les
   proprietes d'un objet de type information sur la sortie
   standard (ecran).
*/

pr_inf (iobj,nb)

/* DECLARATION DES VARIABLES PARAMETRES */

   struct itinf iobj ;          /* donnee */
   int    nb ;                  /* donnee */

{
/* DEBUT DE LA PROCEDURE PR_INF */

   if (nb != 0 )  printf ("%d. \n", nb ) ;
   printf ("    nom         : %s \n", iobj.inom) ;
   printf ("    code        : %d \n", iobj.icode) ;
   printf ("    nature      : %s \n", iobj.inature) ;
   printf ("    crdate      : %s \n", iobj.icrdate) ;
   printf ("    creat       : %d \n", iobj.icreat) ;
   printf ("    presence    : %d \n", iobj.ipresence) ;
   printf ("    codate      : %s \n", iobj.icodate) ;
   printf ("    consult     : %d \n", iobj.iconsult) ;
   printf ("    modate      : %s \n", iobj.imodate) ;
   printf ("    modificat   : %d \n", iobj.imodificat) ;
   printf ("    desc        : %s \n", iobj.idesc) ;
   printf ("    stconf      : %s \n", iobj.istconf) ;
   printf ("    type        : %s \n", iobj.itype) ;
   printf ("    poss_con    : %s \n", iobj.iposs_con) ;
   printf ("    poss_maj    : %s \n", iobj.iposs_maj) ;
   printf ("    exped       : %s \n", iobj.iexped) ;
   printf ("    mots_cles   : %s \n", iobj.imots_cles) ;
   printf ("    ref_stck    : %s \n", iobj.iref_stck) ;
   printf ("    ref_orig    : %s \n", iobj.iref_orig) ;
   printf ("    code_t      : %d \n", iobj.icont_t) ;
   printf ("    code_c      : %d \n\n", iobj.icont_c) ;

   return (0) ;

}  /* FIN DE LA PROCEDURE PR_INF */
```

```
#  include "stdio.h"
## include "lib_c.qc"
## include "lib_v.qc"

/*****************************************************************
******************************************************************
**                        PROCEDURE    CREAT                     **
******************************************************************
******************************************************************/

/* Specification : la procedure creat () procede a la creation
   proprement dite d'un objet dans l'E.R. Cette creation se fait
   au niveau physique c'est-a-dire dans l'SGBD cible utilise.
   Dans ce cas il s'agit de l'SGBD INGRES
*/


creat (obj, code_err)

/* DECLARATION DES VARIABLES PARAMETRES */

   tobject obj ;           /* donnee */
   int  *code_err ;        /* resultat */

{
/* DECLARATION DES VARIABLES LOCALES A LA PROCEDURE */
##  struct ituti     uobj ;
##  struct itgruti   guobj ;
##  struct itenvran  eobj ;
##  struct itarm     aobj ;
##  struct ittir     tobj ;
##  struct itfar     fobj ;
##  struct itpou     pobj ;
##  struct itboi     bobj ;
##  struct itgrinf   giobj ;
##  struct itinf     iobj ;

/* DEBUT DE LA PROCEDURE */

/* affectation d'une partie de l'objet a une variable
   particuliere pour faire l'interface avec Ingres.
   Ingres n'accepte que des structures d'un niveau et
   tous les noms des champs des structures doivent etre
   differents. */

   switch (obj.type_object)
   {
   case sutil      : uobj = obj.o.u ;
                     break ;
   case sgr_util   : guobj = obj.o.gu ;
                     break ;
   case senv_rang : eobj = obj.o.e ;
                     break ;
   case sarm       : aobj = obj.o.a ;
                     break ;
   case stir       : tobj = obj.o.t ;
                     break ;
```

```
   case sfar       : fobj = obj.o.f;
                     break ;
   case spou       : pobj = obj.o.p;
                     break ;
   case sboi       : bobj = obj.o.b;
                     break ;
   case sgr_info   : giobj = obj.o.gi;
                     break ;
   case sinfo      : iobj = obj.o.i;
   }       /* sortie switch obj.type_object */

/* Traitement de chaque type d'objet */

   switch (obj.type_object)

   {
     /************************************************
     *                CREAT    UTILISATEUR            *
     ************************************************/

   case sutil :

## append to utilisateur (nom=uobj.unom,code=uobj.ucode,
##                        pwd=uobj.upwd,crdate=uobj.ucrdate,
##                        creat=uobj.ucreat,perm=uobj.uperm,
##                        groupe=uobj.ugroupe)

## inquire_equel (row_count = "rowcount")

   switch (row_count)
   {
      case 0  : *code_err = BD_0_CREAT ; break ;
      case 1  : *code_err = OK ;
   }            /* sortie du case row_count */

   break ;      /* sortie du case sutil */


     /************************************************
     *            CREAT     GROUPE_UTILISATEUR       *
     ************************************************/

   case sgr_util :

## append to gr_util (nom=guobj.gunom,code=guobj.gucode,
##                    crdate=guobj.gucrdate,creat=guobj.gucreat)

## inquire_equel (row_count = "rowcount")

   switch (row_count)
   {
      case 0  : *code_err = BD_0_CREAT ; break ;
      case 1  : *code_err = OK ;
   }            /* sortie du case row_count */

   break ;      /* sortie du case sgr_util */
```

```
   /*************************************************
   *                CREAT       ENV_RANG             *
   *************************************************/

   case senv_rang :

   /* pas de suppression possible de l'ENV_RANG */

   *code_err = NOT_IMPLEM ;

   break ;      /* sortie du case ENV_RANG */


   /*************************************************
   *                CREAT       POUBELLE             *
   *************************************************/

   case spou :

   /* pas de suppression possible de la POUBELLE */

   *code_err = NOT_IMPLEM ;

   break ;      /* sortie du case spou */


   /*************************************************
   *                CREAT       BOITE                *
   *************************************************/

   case sboi :

## append to boite  (nom=bobj.bnom,code=bobj.bcode,
##                   crdate=bobj.bcrdate,creat=bobj.bcreat,
##                   stconf=bobj.bstconf,poss=bobj.bposs,
##                   type=bobj.btype,desc=bobj.bdesc)

## inquire_equel (row_count = "rowcount")

   switch (row_count)
   {
      case 0  : *code_err = BD_0_CREAT ; break ;
      case 1  : *code_err = OK ;
   }           /* sortie du case row_count */

   break ;      /* sortie du case sboi */


   /*************************************************
   *                CREAT       ARMOIRE              *
   *************************************************/

   case sarm :

## append to armoire  (nom=aobj.anom,code=aobj.acode,
##                     crdate=aobj.acrdate,creat=aobj.acreat,
```

```
##                        stconf=aobj.astconf,poss=aobj.aposs,
##                        type=aobj.atype,desc=aobj.adesc)

## inquire_equel (row_count = "rowcount")

   switch (row_count)
   {
      case 0  : *code_err = BD_0_CREAT ; break ;
      case 1  : *code_err = OK ;
   }            /* sortie du case row_count */

   break ;           /* sortie du case sarm */


     /************************************************
     *                  CREAT       TIROIR           *
     ************************************************/

   case stir :

## append to tiroir  (nom=tobj.tnom,code=tobj.tcode,
##                    crdate=tobj.tcrdate,creat=tobj.tcreat,
##                    stconf=tobj.tstconf,poss=tobj.tposs,
##                    type=tobj.ttype,desc=tobj.tdesc,
##                    code_armoire=tobj.tcont_c)

## inquire_equel (row_count = "rowcount")

   switch (row_count)
   {
      case 0  : *code_err = BD_0_CREAT ; break ;
      case 1  : *code_err = OK ;
   }            /* sortie du case row_count */

   break ;      /* sortie du case stir */


     /************************************************
     *                  CREAT      FARDE             *
     ************************************************/

   case sfar :

## append to farde  (nom=fobj.fnom,code=fobj.fcode,
##                   crdate=fobj.fcrdate,creat=fobj.fcreat,
##                   stconf=fobj.fstconf,poss=fobj.fposs,
##                   type=fobj.ftype,desc=fobj.fdesc,
##                   code_tiroir=fobj.fcont_c)

## inquire_equel (row_count = "rowcount")

   switch (row_count)
   {
      case 0  : *code_err = BD_0_CREAT ; break ;
      case 1  : *code_err = OK ;
   }            /* sortie du case row_count */
```

```
   break;       /* sortie du case sfar */


     /*************************************************
     *           CREAT    GROUPE_INFORMATION          *
     *************************************************/

   case sgr_info :

## append to gr_info (nom=giobj.ginom,code=giobj.gicode,
##                    presence=giobj.gipresence,
##                    crdate=giobj.gicrdate,creat=giobj.gicreat,
##                    stconf=giobj.gistconf,
##                    poss_con=giobj.giposs_con,
##                    poss_maj=giobj.giposs_maj,
##                    type=giobj.gitype,desc=giobj.gidesc,
##                    cont_t=giobj.gicont_t,cont_c=giobj.gicont_c)

## inquire_equel (row_count = "rowcount")

   switch (row_count)
   {
      case 0  : *code_err = BD_0_CREAT ; break ;
      case 1  : *code_err = OK ;
   }           /* sortie du case row_count */

   break ;     /* sortie du case sgr_info */


     /*************************************************
     *                CREAT         INFORMATION        *
     *************************************************/

   case sinfo :

## append to information (nom=iobj.inom,code=iobj.icode,
##                        presence=iobj.ipresence,
##                        nature=iobj.inature,
##                        crdate=iobj.icrdate,creat=iobj.icreat,
##                        codate=iobj.icodate,
##                        consult=iobj.iconsult,
##                        modate=iobj.imodate,
##                        modificat=iobj.imodificat,
##                        stconf=iobj.istconf,
##                        poss_con=iobj.iposs_con,
##                        poss_maj=iobj.iposs_maj,
##                        type=iobj.itype,desc=iobj.idesc,
##                        exped=iobj.iexped,
##                        mots_cles=iobj.imots_cles,
##                        ref_stck=iobj.iref_stck,
##                        ref_orig=iobj.iref_orig,
##                        cont_t=iobj.icont_t,cont_c=iobj.icont_c)

## inquire_equel (row_count = "rowcount")

   switch (row_count)
   {
```

```
        case 0   : *code_err = BD_O_CREAT ; break ;
        case 1   : *code_err = OK ;
    }              /* sortie du case row_count */

    break ;        /* sortie du case sinfo */

    }         /* sortie du switch obj.type_object */

} /* FIN DE LA PROCEDURE CREAT */
```

```
#  include "stdio.h"
## include "lib_c.qc"
## include "lib_v.qc"

/*****************************************************************
******************************************************************
**                        PROCEDURE     SUPPR                     **
******************************************************************
******************************************************************/

/* Specification : la procedure suppr () procede a la suppression
   proprement dite d'un objet dans l'E.R. Cette suppression se
   fait au niveau physique c'est-a-dire dans l'SGBD cible utilise.
   Dans ce cas il s'agit de l'SGBD INGRES
*/


suppr (otype, ocode, code_err)

/* DECLARATION DES VARIABLES PARAMETRES */

   int  otype ;                 /* donnee */
## long ocode ;                 /* donnee */
   int  *code_err ;             /* resultat */

{
/* DEBUT DE LA PROCEDURE */

   /* Traitement du type d'objet particulier */
   switch (otype)

   {
     /*************************************************
     *                SUPPR      UTILISATEUR            *
     *************************************************/

     case sutil :

     /* pas de suppression possible d'un utilisateur */
     *code_err = NOT_IMPLEM ;

     break ;      /* sortie du case sutil */


     /*************************************************
     *                SUPPR         GROUP_UTIL          *
     *************************************************/

     case sgr_util :

     /* pas de suppression possible d'un groupe d'utilisateurs */
     *code_err = NOT_IMPLEM ;

     break ;      /* sortie du case sgr_util */
```

```
      /*************************************************
      *                SUPPR       ENV_RANG             *
      *************************************************/

      case senv_rang :

      /* pas de suppression possible de l'ENV_RANG */
      *code_err = NOT_IMPLEM ;

      break ;      /* sortie du case senv_rang */


      /*************************************************
      *                SUPPR         POUBELLE           *
      *************************************************/

      case spou :

      /* pas de suppression possible de la POUBELLE */
      *code_err = NOT_IMPLEM ;

      break ;      /* sortie du case spou */


      /*************************************************
      *                 SUPPR        BOITE              *
      *************************************************/

   case sboi :

## delete b where b.code = ocode

## inquire_equel (row_count = "rowcount")

   switch (row_count)
   {
      case 0  : *code_err = BD_0_SUPP ; break ;
      case 1  : *code_err = OK ; break ;
      default : *code_err = BD_CONT_INT ;
   }          /* sortie du case row_count */

   break ;      /* sortie du case sboi */


      /*************************************************
      *                SUPPR         ARMOIRE            *
      *************************************************/

   case sarm :

## delete a where a.code = ocode

## inquire_equel (row_count = "rowcount")

   switch (row_count)
   {
```

```
      case 0  : *code_err = BD_0_SUPP ; break ;
      case 1  : *code_err = OK ; break ;
      default : *code_err = BD_CONT_INT ;
   }          /* sortie du case row_count */

   break ;           /* sortie du case sarm */


     /*************************************************
     *                  SUPPR          TIROIR          *
     *************************************************/

   case stir :

## delete t where t.code = ocode

## inquire_equel (row_count = "rowcount")

   switch (row_count)
   {
      case 0  : *code_err = BD_0_SUPP ; break ;
      case 1  : *code_err = OK ; break ;
      default : *code_err = BD_CONT_INT ;
   }          /* sortie du case row_count */

   break ;      /* sortie du case stir */


     /*************************************************
     *                  SUPPR         FARDE            *
     *************************************************/

   case sfar :

## delete f where f.code = ocode

## inquire_equel (row_count = "rowcount")

   switch (row_count)
   {
      case 0  : *code_err = BD_0_SUPP ; break ;
      case 1  : *code_err = OK ; break ;
      default : *code_err = BD_CONT_INT ;
   }          /* sortie du case row_count */

   break;       /* sortie du case sfar */


     /*************************************************
     *           SUPPR   GROUPE_INFORMATION           *
     *************************************************/

   case sgr_info :

## delete gi where gi.code = ocode

## inquire_equel (row_count = "rowcount")
```

```
    switch (row_count)
    {
       case 0  : *code_err = BD_0_SUPP ; break ;
       case 1  : *code_err = OK ; break ;
       default : *code_err = BD_CONT_INT ;
    }          /* sortie du case row_count */

    break ;      /* sortie du case sgr_info */


      /************************************************
      *             SUPPR         INFORMATION          *
      ************************************************/

      case sinfo :

## delete i where i.code = ocode

## inquire_equel (row_count = "rowcount")

    switch (row_count)
    {
       case 0  : *code_err = BD_0_SUPP ; break ;
       case 1  : *code_err = OK ; break ;
       default : *code_err = BD_CONT_INT ;
    }          /* sortie du case row_count */

    break ;      /* sortie du case sinfo */

  }  /* sortie du switch otype */

}    /* FIN DE LA PROCEDURE SUPPR */
```

```
# include "stdio.h"
## include "lib_c.qc"
## include "lib_v.qc"

/*****************************************************************
******************************************************************
**                        PROCEDURE    MODIF                     **
******************************************************************
******************************************************************/

/* Specification : la procedure modif () procede a la modification
   proprement dite d'une ou plusieurs proprietes d'un objet dans
   l'E.R. Cette consultation se fait au niveau physique
   c'est-a-dire dans l'SGBD cible utilise.
   Dans ce cas il s'agit de l'SGBD INGRES
*/


modif (obj, code_err)

/* DECLARATION DES VARIABLES PARAMETRES */

   tobject obj ;            /* donnee */
   int  *code_err ;         /* resultat */

{
/* DECLARATION DES VARIABLES LOCALES A LA PROCEDURE */
##  struct ituti     uobj ;
##  struct itgruti   guobj ;
##  struct itenvran  eobj ;
##  struct itarm     aobj ;
##  struct ittir     tobj ;
##  struct itfar     fobj ;
##  struct itpou     pobj ;
##  struct itboi     bobj ;
##  struct itgrinf   giobj ;
##  struct itinf     iobj ;

/* DEBUT DE LA PROCEDURE */

/* affectation d'une partie de l'objet a une variable
   particuliere pour faire l'interface avec Ingres.
   Ingres n'accepte que des structures d'un niveau et
   tous les noms des champs des structures doivent etre
   differents. */

   switch (obj.type_object)
   {
   case sutil      : uobj = obj.o.u ;
                     break ;
   case sgr_util   : guobj = obj.o.gu ;
                     break ;
   case senv_rang : eobj = obj.o.e ;
                     break ;
   case sarm       : aobj = obj.o.a ;
                     break ;
   case stir       : tobj = obj.o.t ;
```

```
                    break ;
   case sfar      : fobj = obj.o.f;
                    break ;
   case spou      : pobj = obj.o.p;
                    break ;
   case sboi      : bobj = obj.o.b;
                    break ;
   case sgr_info  : giobj = obj.o.gi;
                    break ;
   case sinfo     : iobj = obj.o.i;
   }      /* sortie switch obj.type_object */

/* Traitement de chaque type d'objet */

   switch (obj.type_object)

   {
     /*************************************************
     *                MODIF    UTILISATEUR             *
     *************************************************/

     case sutil :

## replace u (nom=uobj.unom,
##            pwd=uobj.upwd,crdate=uobj.ucrdate,
##            creat=uobj.ucreat,perm=uobj.uperm,
##            groupe=uobj.ugroupe)
## where u.code = uobj.ucode

## inquire_equel (row_count = "rowcount")

   switch (row_count)
   {
      case 0  : *code_err = BD_0_MODIF ; break ;
      case 1  : *code_err = OK ; break ;
      default : *code_err = BD_CONT_INT ;
   }           /* sortie du case row_count */

   break ;     /* sortie du case sutil */


     /*************************************************
     *          MODIF    GROUPE_UTILISATEUR           *
     *************************************************/

     case sgr_util :

## replace gu (nom=guobj.gunom,
##             crdate=guobj.gucrdate,creat=guobj.gucreat)
## where gu.code = guobj.gucode

## inquire_equel (row_count = "rowcount")

   switch (row_count)
   {
      case 0  : *code_err = BD_0_MODIF ; break ;
      case 1  : *code_err = OK ; break ;
```

```
      default : *code_err = BD_CONT_INT ;
   }            /* sortie du case row_count */

   break ;      /* sortie du case sgr_util */


     /*************************************************
     *                MODIF      ENV_RANG              *
     *************************************************/

     case senv_rang :

## replace e  (nom=eobj.enom,
##             crdate=eobj.ecrdate,creat=eobj.ecreat,
##             stconf=eobj.estconf,poss=eobj.eposs,
##             desc=eobj.edesc)
## where e.code = eobj.ecode

## inquire_equel (row_count = "rowcount")

   switch (row_count)
   {
      case 0  : *code_err = BD_0_MODIF ; break ;
      case 1  : *code_err = OK ; break ;
      default : *code_err = BD_CONT_INT ;
   }            /* sortie du case row_count */

   break ;      /* sortie du case ENV_RANG */


     /*************************************************
     *                MODIF       POUBELLE             *
     *************************************************/

     case spou :

## replace p  (nom=pobj.pnom,
##             crdate=pobj.pcrdate,creat=pobj.pcreat,
##             stconf=pobj.pstconf,poss=pobj.pposs,
##             desc=pobj.pdesc)
## where p.code = pobj.pcode

## inquire_equel (row_count = "rowcount")

   switch (row_count)
   {
      case 0  : *code_err = BD_0_MODIF ; break ;
      case 1  : *code_err = OK ; break ;
      default : *code_err = BD_CONT_INT ;
   }            /* sortie du case row_count */

   break ;      /* sortie du case spou */
```

```
      /*************************************************
      *                  MODIF      BOITE               *
      *************************************************/

      case sboi :

## replace b  (nom=bobj.bnom,
##             crdate=bobj.bcrdate,creat=bobj.bcreat,
##             stconf=bobj.bstconf,poss=bobj.bposs,
##             type=bobj.btype,desc=bobj.bdesc)
## where b.code = bobj.bcode

## inquire_equel (row_count = "rowcount")

   switch (row_count)
   {
      case 0  : *code_err = BD_0_MODIF ; break ;
      case 1  : *code_err = OK ; break ;
      default : *code_err = BD_CONT_INT ;
   }          /* sortie du case row_count */

   break ;    /* sortie du case sboi */


      /*************************************************
      *                  MODIF      ARMOIRE             *
      *************************************************/

      case sarm :

## replace a  (nom=aobj.anom,
##             crdate=aobj.acrdate,creat=aobj.acreat,
##             stconf=aobj.astconf,poss=aobj.aposs,
##             type=aobj.atype,desc=aobj.adesc)
## where a.code = aobj.acode

## inquire_equel (row_count = "rowcount")

   switch (row_count)
   {
      case 0  : *code_err = BD_0_MODIF ; break ;
      case 1  : *code_err = OK ; break ;
      default : *code_err = BD_CONT_INT ;
   }          /* sortie du case row_count */

   break ;         /* sortie du case sarm */


      /*************************************************
      *                  MODIF      TIROIR              *
      *************************************************/

      case stir :

## replace t  (nom=tobj.tnom,
##             crdate=tobj.tcrdate,creat=tobj.tcreat,
```

```
##              stconf=tobj.tstconf,poss=tobj.tposs,
##              type=tobj.ttype,desc=tobj.tdesc,
##              code_armoire=tobj.tcont_c)
## where t.code = tobj.tcode

## inquire_equel (row_count = "rowcount")

   switch (row_count)
   {
      case 0  : *code_err = BD_0_MODIF ; break ;
      case 1  : *code_err = OK ; break ;
      default : *code_err = BD_CONT_INT ;
   }           /* sortie du case row_count */

   break ;     /* sortie du case stir */


     /*************************************************
     *                 MODIF      FARDE                *
     *************************************************/

     case sfar :

## replace f  (nom=fobj.fnom,
##             crdate=fobj.fcrdate,creat=fobj.fcreat,
##             stconf=fobj.fstconf,poss=fobj.fposs,
##             type=fobj.ftype,desc=fobj.fdesc,
##             code_tiroir=fobj.fcont_c)
## where f.code = fobj.fcode

## inquire_equel (row_count = "rowcount")

   switch (row_count)
   {
      case 0  : *code_err = BD_0_MODIF ; break ;
      case 1  : *code_err = OK ; break ;
      default : *code_err = BD_CONT_INT ;
   }           /* sortie du case row_count */

   break;      /* sortie du case sfar */


     /*************************************************
     *           MODIF   GROUPE_INFORMATION           *
     *************************************************/

     case sgr_info :

## replace gi (nom=giobj.ginom,
##             presence=giobj.gipresence,
##             crdate=giobj.gicrdate,creat=giobj.gicreat,
##             stconf=giobj.gistconf,poss_con=giobj.giposs_con,
##             poss_maj=giobj.giposs_maj,
##             type=giobj.gitype,desc=giobj.gidesc,
##             cont_t=giobj.gicont_t,cont_c=giobj.gicont_c)
## where gi.code = giobj.gicode
```

```
## inquire_equel (row_count = "rowcount")

   switch (row_count)
   {
      case 0  : *code_err = BD_0_MODIF ; break ;
      case 1  : *code_err = OK ; break ;
      default : *code_err = BD_CONT_INT ;
   }            /* sortie du case row_count */

   break ;      /* sortie du case sgr_info */


     /*************************************************
     *                MODIF     INFORMATION            *
     *************************************************/

     case sinfo :

## replace i (nom=iobj.inom,
##            presence=iobj.ipresence,nature=iobj.inature,
##            crdate=iobj.icrdate,creat=iobj.icreat,
##            codate=iobj.icodate,consult=iobj.iconsult,
##            modate=iobj.imodate,modificat=iobj.imodificat,
##            stconf=iobj.istconf,
##            poss_con=iobj.iposs_con,
##            poss_maj=iobj.iposs_maj,
##            type=iobj.itype,desc=iobj.idesc,
##            exped=iobj.iexped,mots_cles=iobj.imots_cles,
##            ref_stck=iobj.iref_stck,
##            ref_orig=iobj.iref_orig,
##            cont_t=iobj.icont_t,cont_c=iobj.icont_c)
## where i.code = iobj.icode

## inquire_equel (row_count = "rowcount")

   switch (row_count)
   {
      case 0  : *code_err = BD_0_MODIF ; break ;
      case 1  : *code_err = OK ; break ;
      default : *code_err = BD_CONT_INT ;
   }            /* sortie du case row_count */

   break ;      /* sortie du case sinfo */

   }       /* sortie du switch obj.type_object */

} /* FIN DE LA PROCEDURE MODIF */
```

```
#  include "stdio.h"
## include "lib_c.qc"
## include "lib_v.qc"

/*****************************************************************
******************************************************************
**                      PROCEDURE        CONSUL                  **
******************************************************************
******************************************************************/

/* Specification : la procedure consul () procede a la
   consultation proprement dite d'un objet dans l'E.R.
   Cette consultation se fait au niveau physique
   c'est-a-dire dans l'SGBD cible utilise.
   Dans ce cas il s'agit de l'SGBD INGRES.
*/


consul (obj, mode, code_err)

/* DECLARATION DES VARIABLES PARAMETRES */

    tobject *obj ;            /* donnee et resultat */
    char    mode ;            /* donnee */
    int     *code_err ;       /* resultat */

/* Remarque : pour le parametre *obj, seul de type, le code et la
   presence de l'objet suffisent pour l'ouverture. Pour la ferme-
   ture, il faut egalement la date de consultation et le consult.
*/

{
/* DECLARATION DES VARIABLES LOCALES A LA PROCEDURE */
##  struct itgrinf  giobj ;
##  struct itinf    iobj ;

/* DEBUT DE LA PROCEDURE */

/* affectation d'une partie de l'objet a une variable
   particuliere pour faire l'interface avec Ingres.
   Ingres n'accepte que des structures d'un niveau et
   tous les noms des champs des structures doivent etre
   differents. */

   switch (obj->type_object)
   {
   case sgr_info  : giobj = obj->o.gi;  break ;
   case sinfo     : iobj  = obj->o.i ;  break ;
   default        : *code_err = OP_TOB ;
                    return (0) ;  /* sortie prematuree de consul
                                     pour cause d'erreur */
   }      /* sortie switch obj.type_object */

   /* Traitement du type d'objet particulier */
   switch (obj->type_object)

   {
```

```
      /*************************************************
      *           CONSUL     GROUPE_INFORMATION        *
      *************************************************/

      case sgr_info :

            switch (mode)
            {
            case OUVERTURE :

## replace gi (presence=giobj.gipresence)
## where gi.code = giobj.gicode

## inquire_equel (row_count = "rowcount")

      switch (row_count)
      {
         case 0  : *code_err = BD_OU_CONS ; break ;
         case 1  : *code_err = OK ; break ;
         default : BD_CONT_INT ;
      }            /* sortie du case row_count */

            break ; /* sortie du case OUVERTURE */

            case FERMETURE :

## replace gi (presence=giobj.gipresence)
## where gi.code = giobj.gicode

## inquire_equel (row_count = "rowcount")

      switch (row_count)
      {
         case 0  : *code_err = BD_OF_CONS ; break ;
         case 1  : *code_err = OK ; break ;
         default : BD_CONT_INT ;
      }            /* sortie du case row_count */

            }   /* sortie du switch mode */

      break ;     /* sortie du case sgr_info */


      /*************************************************
      *              CONSUL      INFORMATION           *
      *************************************************/

      case sinfo :

            switch (mode)
            {
            case OUVERTURE :

## replace i (presence=iobj.ipresence)
## where i.code=iobj.icode

## inquire_equel (row_count = "rowcount")
```

```
   switch (row_count)
   {
      case 0  : *code_err = BD_OU_CONS ; break ;
      case 1  : *code_err = OK ; break ;
      default : BD_CONT_INT ;
   }            /* sortie du case row_count */

         break ;  /* sortie du case OUVERTURE */

         case FERMETURE :

## replace i (presence=iobj.ipresence,
##             codate=iobj.icodate, consult=iobj.iconsult)
## where i.code=iobj.icode

## inquire_equel (row_count = "rowcount")

   switch (row_count)
   {
      case 0  : *code_err = BD_OF_CONS ; break ;
      case 1  : *code_err = OK ; break ;
      default : BD_CONT_INT ;
   }            /* sortie du case row_count */

         }  /* sortie du switch mode */

   }     /* sortie du switch obj.type_object */

} /* FIN DE LA PROCEDURE CONSUL */
```

```
#  include "stdio.h"
## include "lib_c.qc"
## include "lib_v.qc"

/*****************************************************************
******************************************************************
**                        PROCEDURE    MAJ                      **
******************************************************************
******************************************************************/

/* Specification : la procedure maj () procede a la mise a jour
   proprement dite d'un objet dans l'E.R. Cette mise a jour se
   fait au niveau physique c'est-a-dire dans l'SGBD cible utilise.
   Dans ce cas il s'agit de l'SGBD INGRES.
*/


maj (obj, mode, code_err)

/* DECLARATION DES VARIABLES PARAMETRES */

    tobject *obj ;            /* donnee et resultat */
    char    mode ;            /* donnee */
    int     *code_err ;       /* resultat */

/* Remarque : pour le parametre *obj, seul de type, le code et la
   presence de l'objet suffisent pour l'ouverture. Pour la
   fermeture, il faut egalement la date de mise a jour et le
   modificat */

{
/* DECLARATION DES VARIABLES LOCALES A LA PROCEDURE */
##  struct itgrinf  giobj ;
##  struct itinf    iobj ;

/* DEBUT DE LA PROCEDURE */

/* affectation d'une partie de l'objet a une variable
   particuliere pour faire l'interface avec Ingres.
   Ingres n'accepte que des structures d'un niveau et
   tous les noms des champs des structures doivent etre
   differents. */

   switch (obj->type_object)
   {
   case sgr_info  : giobj = obj->o.gi;  break ;
   case sinfo     : iobj  = obj->o.i ;  break ;
   default        : *code_err = OP_TOB ;
                    return (0) ;  /* sortie prematuree de maj
                                     pour cause d'erreur */
   }      /* sortie switch obj.type_object */

   /* Traitement du type d'objet particulier */
   switch (obj->type_object)
```

```
    {
      /*************************************************
      *              MAJ     GROUPE_INFORMATION        *
      *************************************************/

      case sgr_info :

            switch (mode)
            {
            case OUVERTURE :

## replace gi (presence=giobj.gipresence)
## where gi.code = giobj.gicode

## inquire_equel (row_count = "rowcount")

      switch (row_count)
      {
         case 0  : *code_err = BD_OU_CONS ; break ;
         case 1  : *code_err = OK ; break ;
         default : BD_CONT_INT ;
      }            /* sortie du case row_count */

            break ; /* sortie du case OUVERTURE */

            case FERMETURE :

## replace gi (presence=giobj.gipresence)
## where gi.code = giobj.gicode

## inquire_equel (row_count = "rowcount")

      switch (row_count)
      {
         case 0  : *code_err = BD_OF_CONS ; break ;
         case 1  : *code_err = OK ; break ;
         default : BD_CONT_INT ;
      }            /* sortie du case row_count */

            }   /* sortie du switch mode */

      break ;     /* sortie du case sgr_info */


      /*************************************************
      *                MAJ     INFORMATION              *
      *************************************************/

      case sinfo :

            switch (mode)
            {
            case OUVERTURE :

## replace i (presence=iobj.ipresence)
## where i.code=iobj.icode
```

```
## inquire_equel (row_count = "rowcount")

    switch (row_count)
    {
       case 0  : *code_err = BD_OU_CONS ; break ;
       case 1  : *code_err = OK ; break ;
       default : BD_CONT_INT ;
    }            /* sortie du case row_count */

          break ;  /* sortie du case OUVERTURE */

          case FERMETURE :

## replace i (presence=iobj.ipresence,
##             codate=iobj.icodate, consult=iobj.iconsult)
## where i.code=iobj.icode

## inquire_equel (row_count = "rowcount")

    switch (row_count)
    {
       case 0  : *code_err = BD_OF_CONS ; break ;
       case 1  : *code_err = OK ; break ;
       default : BD_CONT_INT ;
    }            /* sortie du case row_count */

          }  /* sortie du switch mode */

    }     /* sortie du switch obj.type_object */

} /* FIN DE LA PROCEDURE MAJ */
```

```
#  include "stdio.h"
## include "lib_c.qc"
## include "lib_v.qc"


/*****************************************************************
******************************************************************
**                      PROCEDURE    CONTEN                      **
******************************************************************
******************************************************************/

/* Specification : la procedure conten () procede a la
   verification proprement dite du contenu d'un objet dans
   l'E.R. Cette operation se fait au niveau physique
   c'est-a-dire dans l'SGBD cible utilise. Elle renseigne
   le nombre d'objet qui sont contenus dans un objet.
   Ainsi, une armoire contient des tiroirs, une farde contient
   des informations et des groupes d'informations.
   Dans ce cas il s'agit de l'SGBD INGRES
*/


conten (obj, code_err)

/* DECLARATION DES VARIABLES PARAMETRES */

    tobject   obj ;          /* donnee */
    int  *code_err ;         /* resultat */


{
/* DECLARATION DES VARIABLES LOCALES A LA PROCEDURE */
##  struct ituti     uobj ;
##  struct itgruti   guobj ;
##  struct itenvran  eobj ;
##  struct itarm     aobj ;
##  struct ittir     tobj ;
##  struct itfar     fobj ;
##  struct itpou     pobj ;
##  struct itboi     bobj ;
##  struct itgrinf   giobj ;
##  struct itinf     iobj ;

## long vcode ;
   int sum_row = 0 ;

/* DEBUT DE LA PROCEDURE */
/* affectation d'une partie de l'objet a une variable
   particuliere pour faire l'interface avec Ingres.
   Ingres n'accepte que des structures d'un niveau et
   tous les noms des champs des structures doivent etre
   differents. */

   switch (obj.type_object)
   {
   case sutil      : uobj = obj.o.u ;
                     break ;
```

```
   case sgr_util   : guobj = obj.o.gu ;
                     break ;
   case senv_rang : eobj = obj.o.e ;
                     break ;
   case sarm       : aobj = obj.o.a ;
                     break ;
   case stir       : tobj = obj.o.t ;
                     break ;
   case sfar       : fobj = obj.o.f;
                     break ;
   case spou       : pobj = obj.o.p;
                     break ;
   case sboi       : bobj = obj.o.b;
                     break ;
   case sgr_info   : giobj = obj.o.gi;
                     break ;
   case sinfo      : iobj = obj.o.i;
   }       /* sortie switch obj.type_object */

/* Traitement du type d'objet particulier */

   switch (obj.type_object)

   {
     /*************************************************
     *                CONTEN    UTILISATEUR            *
     *************************************************/

     case sutil :

     /* cette operation ne ne s'applique pas sur un utilisateur */

     return (0) ;

     break ;      /* sortie du case sutil */


     /*************************************************
     *                CONTEN       GROUP_UTIL          *
     *************************************************/

     case sgr_util :

    /* cette operation ne s'applique pas sur un group_util */

     return (0) ;

     break ;      /* sortie du case sgr_util */


     /*************************************************
     *                CONTEN       ENV_RANG            *
     *************************************************/

     case senv_rang :
```

```
## retrieve (vcode=u.code)
##     {
##     }

## inquire_equel (row_count = "rowcount")

   sum_row += row_count ;

## retrieve (vcode=gu.code)
##     {
##     }

## inquire_equel (row_count = "rowcount")

   sum_row += row_count ;

## retrieve (vcode=e.code)
##     {
##     }

## inquire_equel (row_count = "rowcount")

   sum_row += row_count ;

## retrieve (vcode=a.code)
##     {
##     }

## inquire_equel (row_count = "rowcount")

   sum_row += row_count ;

## retrieve (vcode=p.code)
##     {
##     }

## inquire_equel (row_count = "rowcount")

   sum_row += row_count ;

## retrieve (vcode=b.code)
##     {
##     }

## inquire_equel (row_count = "rowcount")

   sum_row += row_count ;

   return (sum_row) ;

   break ;      /* sortie du case senv_rang */
```

```
      /*************************************************
      *                CONTEN        POUBELLE           *
      *************************************************/

      case spou :

## retrieve (vcode=gi.code)
## where gi.cont_c = pobj.pcode
##     (
##     )

## inquire_equel (row_count = "rowcount")

   sum_row += row_count ;

## retrieve (vcode=i.code)
## where i.cont_c = pobj.pcode
##     (
##     )

## inquire_equel (row_count = "rowcount")

   sum_row += row_count ;

   return (sum_row) ;

   break ;      /* sortie du case spou */


      /*************************************************
      *                CONTEN         BOITE              *
      *************************************************/

      case sboi :

## retrieve (vcode=gi.code)
## where gi.cont_c = bobj.bcode
##     (
##     )

## inquire_equel (row_count = "rowcount")

   sum_row += row_count ;

## retrieve (vcode=i.code)
## where i.cont_c = bobj.bcode
##     (
##     )

## inquire_equel (row_count = "rowcount")

   sum_row += row_count ;

   return (sum_row) ;

   break ;      /* sortie du case sboi */
```

```
      /*************************************************
      *                  CONTEN      ARMOIRE              *
      *************************************************/

      case sarm :

## retrieve (vcode=t.code)
## where t.code_armoire = aobj.acode
##     (
##     )

## inquire_equel (row_count = "rowcount")

   sum_row += row_count ;

   return (sum_row) ;

   break ;           /* sortie du case sarm */


      /*************************************************
      *                  CONTEN      TIROIR               *
      *************************************************/

      case stir :

## retrieve (vcode=f.code)
## where f.code_tiroir = tobj.tcode
##     (
##     )

## inquire_equel (row_count = "rowcount")

   sum_row += row_count ;

## retrieve (vcode=gi.code)
## where gi.cont_c = tobj.tcode
##     (
##     )

## inquire_equel (row_count = "rowcount")

   sum_row += row_count ;

## retrieve (vcode=i.code)
## where i.cont_c = tobj.tcode
##     (
##     )

## inquire_equel (row_count = "rowcount")

   sum_row += row_count ;

   return (sum_row) ;
```

```
   break ;      /* sortie du case stir */

     /*************************************************
     *                CONTEN     FARDE                 *
     *************************************************/

     case sfar :

## retrieve (vcode=gi.code)
## where gi.cont_c = fobj.fcode
##     {
##     }

## inquire_equel (row_count = "rowcount")

   sum_row += row_count ;

## retrieve (vcode=i.code)
## where i.cont_c = fobj.fcode
##     {
##     }

## inquire_equel (row_count = "rowcount")

   sum_row += row_count ;

   return (sum_row) ;

   break;       /* sortie du case sfar */

     /*************************************************
     *           CONTEN   GROUPE_INFORMATION          *
     *************************************************/

     case sgr_info :

## retrieve (vcode=i.code)
## where i.cont_c = giobj.gicode
##     {
##     }

## inquire_equel (row_count = "rowcount")

   sum_row += row_count ;

   return (sum_row) ;

   break ;      /* sortie du case sgr_info */

     /*************************************************
     *               CONTEN        INFORMATION         *
     *************************************************/

     case sinfo :
```

```
      /* une information ne contient pas d'autres objets */
      return (0) ;
      break ;       /* sortie du case sinfo */
   }  /* sortie du switch obj.type_object */
} /* FIN DE LA PROCEDURE CONTEN */
```

```
# include "stdio.h"
## include "lib_c.qc"
## include "lib_v.qc"

/*****************************************************************
******************************************************************
**                        PROCEDURE      RECH                    **
******************************************************************
******************************************************************/

/* Specification : la procedure rech () procede a la recherche
   proprement dite d'objets dans l'E.R. Cette creation se fait
   au niveau physique c'est-a-dire dans l'SGBD cible utilise.
   Dans ce cas il s'agit de l'SGBD INGRES
*/


rech (otype, cond, res, code_err)

/* DECLARATION DES VARIABLES PARAMETRES */

   int  otype ;              /* donnee */
## char cond[] ;             /* donnee */
   list *res ;               /* resulat */
   int  *code_err ;          /* resultat */

{
/* DECLARATION DES VARIABLES LOCALES A LA PROCEDURE */
## int rcode ;               /* resultat de la requete */
   list *courant ;

/* DEBUT DE LA PROCEDURE */

/* Traitement de chaque type d'objet */

   switch (otype)

   {
     /*************************************************
     *                  RECH    UTILISATEUR            *
     *************************************************/

     case sutil :

   courant = (list *) malloc (sizeof (list)) ;
   res->suiv = courant ;

## retrieve (rcode = u.code)
## where cond
## {
   courant->val = rcode ;
   courant->suiv = (list *) malloc (sizeof (list)) ;
   courant = courant->suiv ;
   courant->suiv = NULL ;
## }

## inquire_equel (row_count = "rowcount")
```

```
   return (row_count) ;

   break ;      /* sortie du case sutil */


     /*************************************************
     *              RECH    GROUPE_UTILISATEUR        *
     *************************************************/

     case sgr_util :

   courant = (list *) malloc (sizeof (list)) ;
   res->suiv = courant ;

## retrieve (rcode = gu.code)
## where cond
## {
   courant->val = rcode ;
   courant->suiv = (list *) malloc (sizeof (list)) ;
   courant = courant->suiv ;
   courant->suiv = NULL ;
## }

## inquire_equel (row_count = "rowcount")
   return (row_count) ;

   break ;      /* sortie du case sgr_util */


     /*************************************************
     *                    RECH   ENV_RANG              *
     *************************************************/

     case senv_rang :

   courant = (list *) malloc (sizeof (list)) ;
   res->suiv = courant ;

## retrieve (rcode = e.code)
## where cond
## {
   courant->val = rcode ;
   courant->suiv = (list *) malloc (sizeof (list)) ;
   courant = courant->suiv ;
   courant->suiv = NULL ;
## }

## inquire_equel (row_count = "rowcount")
   return (row_count) ;

   break ;      /* sortie du case ENV_RANG */


     /*************************************************
     *                    RECH     POUBELLE            *
     *************************************************/
```

```
     case spou :

   courant = (list *) malloc (sizeof (list)) ;
   res->suiv = courant ;

## retrieve (rcode = p.code)
## where cond
## (
   courant->val = rcode ;
   courant->suiv = (list *) malloc (sizeof (list)) ;
   courant = courant->suiv ;
   courant->suiv = NULL ;
## )

## inquire_equel (row_count = "rowcount")
   return (row_count) ;

   break ;      /* sortie du case spou */


     /*************************************************
     *                    RECH     BOITE                *
     *************************************************/

     case sboi :

   courant = (list *) malloc (sizeof (list)) ;
   res->suiv = courant ;

## retrieve (rcode = b.code)
## where cond
## (
   courant->val = rcode ;
   courant->suiv = (list *) malloc (sizeof (list)) ;
   courant = courant->suiv ;
   courant->suiv = NULL ;
## )

## inquire_equel (row_count = "rowcount")
   return (row_count) ;

   break ;      /* sortie du case sboi */


     /*************************************************
     *                    RECH     ARMOIRE              *
     *************************************************/

     case sarm :

   courant = (list *) malloc (sizeof (list)) ;
   res->suiv = courant ;

## retrieve (rcode = a.code)
## where cond
## (
   courant->val = rcode ;
```

```
   courant->suiv = (list *) malloc (sizeof (list)) ;
   courant = courant->suiv ;
   courant->suiv = NULL ;
## }

## inquire_equel (row_count = "rowcount")
   return (row_count) ;

   break ;          /* sortie du case sarm */


     /*************************************************
     *                    RECH     TIROIR                *
     *************************************************/

     case stir :

   courant = (list *) malloc (sizeof (list)) ;
   res->suiv = courant ;

## retrieve (rcode = t.code)
## where cond
## {
   courant->val = rcode ;
   courant->suiv = (list *) malloc (sizeof (list)) ;
   courant = courant->suiv ;
   courant->suiv = NULL ;
## }

## inquire_equel (row_count = "rowcount")
   return (row_count) ;

   break ;      /* sortie du case stir */


     /*************************************************
     *                    RECH     FARDE                 *
     *************************************************/

     case sfar :

   courant = (list *) malloc (sizeof (list)) ;
   res->suiv = courant ;

## retrieve (rcode = f.code)
## where cond
## {
   courant->val = rcode ;
   courant->suiv = (list *) malloc (sizeof (list)) ;
   courant = courant->suiv ;
   courant->suiv = NULL ;
## }

## inquire_equel (row_count = "rowcount")
   return (row_count) ;

   break;       /* sortie du case sfar */
```

```
      /*************************************************
      *              RECH   GROUPE_INFORMATION          *
      *************************************************/

        case sgr_info :

     courant = (list *) malloc (sizeof (list)) ;
     res->suiv = courant ;

  ## retrieve (rcode = gi.code)
  ## where cond
  ## (
     courant->val = rcode ;
     courant->suiv = (list *) malloc (sizeof (list)) ;
     courant = courant->suiv ;
     courant->suiv = NULL ;
  ## )

  ## inquire_equel (row_count = "rowcount")
     return (row_count) ;

     break ;      /* sortie du case sgr_info */


      /*************************************************
      *                RECH      INFORMATION            *
      *************************************************/

        case sinfo :

     courant = (list *) malloc (sizeof (list)) ;
     res->suiv = courant ;

  ## retrieve (rcode = i.code)
  ## where cond
  ## (
     courant->val = rcode ;
     courant->suiv = (list *) malloc (sizeof (list)) ;
     courant = courant->suiv ;
     courant->suiv = NULL ;
  ## )

  ## inquire_equel (row_count = "rowcount")
     return (row_count) ;

     break ;      /* sortie du case sinfo */

     }       /* sortie du switch obj.type_object */

  } /* FIN DE LA PROCEDURE RECH */
```

```
#  include "creat.c"


/*****************************************************************
*****************************************************************
**                      PROCEDURE     CREATION                   **
*****************************************************************
*****************************************************************/

/* Specification : la procedure creation () procede a la creation
   d'un nouvel objet dans l'E.R. a la demande d'un utilisateur.
   Cette procedure verifie un certain nombre de conditions avant
   de proceder a la creation proprement dite de l'objet.
   A son retour, elle confirme la creation ou renseigne sur
   la cause de la non creation.
*/


creation (util, date_jour, obj, code_ret)

/* DECLARATION DES VARIABLES PARAMETRES */

struct  user  util ;                    /* donnee */
char    date_jour[26] ;                 /* donnee */
tobject *obj ;                          /* donnee et resultat */
int     *code_ret ;                     /* resultat */

{
/* DEBUT DE LA PROCEDURE */

   /* initialisation de la variable *code_ret */
   *code_ret = OK ;

   /* Verification de la permission de l'utilisateur */
   if (util.per != ECRITURE)
   {
   *code_ret = AC_ECRIT ;
   return (0) ;    /* sortie prematuree de creation pour manque
                      de permission de l'utilisateur appelant */
   }

   /* Traitement du type d'objet particulier */
   switch (obj->type_object)
   {

     /*************************************************
     *                CREATION UTILISATEUR             *
     *************************************************/

     case sutil :

     /* Verifier que l'utilisateur appelant est l'adm. */
     if (util.cod != cadm)
     {  *code_ret = AC_ADM ;
        break ;     /* sortie prematuree du case sutil */
     }
```

```
/* Pas de verification possible du nom */

/* Generation automatique du code */
gen_compt (&(obj->o.u.ucode), code_ret) ;
if (*code_ret != OK)        /* code_ret est une adresse & */
break ;      /* sortie prematuree du case sutil */

/* Pas de verification possible du mot de passe */

/* Affectation de la date du jour --> crdate */
strcpy (obj->o.u.ucrdate,date_jour) ;

/* Affectation du code utilisateur --> creat */
obj->o.u.ucreat = util.cod ;

/* creation proprement dite */
creat (*obj, code_ret) ;

break ;      /* sortie du case sutil */


/*************************************************
*            CREATION        GR_UTIL             *
*************************************************/

case sgr_util :

/* Verifier que l'utilisateur appelant est l'adm. */
if (util.cod != cadm)
{  *code_ret = AC_ADM ;
   break ;
}

/* Pas de verification possible du nom */

/* Generation automatique du code */
gen_compt (&(obj->o.gu.gucode), code_ret) ;
if (*code_ret != OK)
break ;      /* sortie prematuree du case sgr_util */

/* Affectation de la date du jour --> crdate */
strcpy (obj->o.gu.gucrdate, date_jour) ;

/* Affectation du code utilisateur --> creat */
obj->o.gu.gucreat = util.cod ;

/* creation proprement dite */
creat (*obj, code_ret) ;

break ;      /* sortie du case sgr_util */


/*************************************************
*               CREATION   ENV_RANG               *
*************************************************/

case senv_rang :
```

```
/* pas de creation possible de l'ENV_RANG */
*code_ret = NOT_IMPLEM ;

break ;      /* sortie du case senv_rang */


/*************************************************
*                CREATION     POUBELLE            *
*************************************************/

case spou :

/* pas de creation possible de POUBELLE */
*code_ret = NOT_IMPLEM ;

break ;      /* sortie du case spou */


/*************************************************
*                CREATION     BOITE               *
*************************************************/

case sboi :

/* Verifier que l'utilisateur appelant est l'adm. */
if (util.cod != cadm)
{  *code_ret = AC_ADM ;
   break ;
}

/* Pas de verification possible du nom */

/* Generation automatique du code */
gen_compt ( &(obj->o.b.bcode), code_ret) ;
if (*code_ret != OK)
break ;      /* sortie prematuree du case sboi */

/* Affectation de la date du jour --> crdate */
strcpy (obj->o.b.bcrdate,date_jour) ;

/* Affectation du code utilisateur --> creat */
obj->o.b.bcreat = util.cod ;

/* Pas de verification possible du stconf */

/* Pas de verification possible du type */

/* Pas de verification possible de la desc */

/* Pas de verification possible des poss */

/* creation proprement dite */
creat (*obj, code_ret) ;

break ;      /* sortie du case sboi */
```

```
/*************************************************
*                CREATION     ARMOIRE             *
*************************************************/

case sarm :

/* Verifier que l'utilisateur appelant est l'adm. */
if (util.cod != cadm)
{  *code_ret = AC_ADM ;
   break ;
}

/* Pas de verification possible du nom */

/* Generation automatique du code */
gen_compt ( &(obj->o.a.acode), code_ret) ;
if (*code_ret != OK)
break ;      /* sortie prematuree du case sarm */

/* Affectation de la date du jour --> crdate */
strcpy (obj->o.a.acrdate,date_jour) ;

/* Affectation du code utilisateur --> creat */
obj->o.a.acreat = util.cod ;

/* Pas de verification possible du stconf */

/* Pas de verification possible du type */

/* Pas de verification possible de la desc */

/* Pas de verification possible des poss */

/* creation proprement dite */
creat (*obj, code_ret) ;

break ;          /* sortie du case sarm */


/*************************************************
*                CREATION     TIROIR              *
*************************************************/

case stir :

/* Verifier que l'utilisateur appelant est l'adm. */
if (util.cod != cadm)
{  *code_ret = AC_ADM ;
   break ;          /* sortie prematuree du case stir */
}

/* Verifier si l'armoire, a laquelle appartiendra le tiroir,
   existe */
exist (sarm, obj->o.t.tcont_c, code_ret) ;
if (*code_ret != OK)
break ;          /* sortie prematuree du case stir */
```

```
/* Verifier si l'utilisateur a acces a cette armoire */
acces (sarm, obj->o.t.tcont_c, util.group, code_ret) ;
if (*code_ret != OK)
break ;          /* sortie prematuree du case stir */

/* Pas de verification possible du nom */

/* Generation automatique du code */
gen_compt ( &(obj->o.t.tcode), code_ret) ;
if (*code_ret != OK)
break ;      /* sortie prematuree du case stir */

/* Affectation de la date du jour --> crdate */
strcpy (obj->o.t.tcrdate,date_jour) ;

/* Affectation du code utilisateur --> creat */
obj->o.t.tcreat = util.cod ;

/* Pas de verification possible du stconf */

/* Pas de verification possible du type */

/* Pas de verification possible de la desc */

/* Pas de verification possible des poss */

/* creation proprement dite */
creat (*obj, code_ret) ;

break ;      /* sortie du case TIROIR */


/*************************************************
*                CREATION     FARDE               *
*************************************************/

case sfar :

/* Verifier si le tiroir, auquel appartiendra la farde,
   existe */
exist (stir, obj->o.f.fcont_c, code_ret) ;
if (*code_ret != OK)
break ;          /* sortie prematuree du case sfar */

/* Verifier si l'utilisateur a acces a ce tiroir */
acces (stir, obj->o.f.fcont_c, util.group, code_ret) ;
if (*code_ret != OK)
break ;          /* sortie prematuree du case sfar */

/* Pas de verification possible du nom */

/* Generation automatique du code */
gen_compt ( &(obj->o.f.fcode), code_ret) ;
if (*code_ret != OK)
break ;      /* sortie prematuree du case sfar */
```

```
/* Affectation de la date du jour --> crdate */
strcpy (obj->o.f.fcrdate,date_jour) ;

/* Affectation du code utilisateur --> creat */
obj->o.f.fcreat = util.cod ;

/* Pas de verification possible du stconf */

/* Pas de verification possible du type */

/* Pas de verification possible de la desc */

/* Pas de verification possible des poss */

/* creation proprement dite */
creat (*obj, code_ret) ;

break;       /* sortie du case FARDE */


/************************************************
*         CREATION     GROUPE_INFORMATION       *
************************************************/

case sgr_info :

/* Verifier si l'objet, auquel appartiendra le groupe
   d'informations , existe */
exist (obj->o.gi.gicont_t, obj->o.gi.gicont_c, code_ret) ;
if (*code_ret != OK)
break ;          /* sortie prematuree du case sgr_info */

/* Verifier si l'utilisateur a acces a cet objet */
acces (obj->o.gi.gicont_t, obj->o.gi.gicont_c, util.group,
       code_ret) ;
if (*code_ret != OK)
break ;          /* sortie prematuree du case sgr_info */

/* Pas de verification possible du nom */

/* Generation automatique du code */
gen_compt ( &(obj->o.gi.gicode), code_ret) ;
if (*code_ret != OK)
break ;      /* sortie prematuree du case sgr_info */

/* Affectation de la date du jour --> crdate */
strcpy (obj->o.gi.gicrdate,date_jour) ;

/* Affectation du code utilisateur --> creat */
obj->o.gi.gicreat = util.cod ;

/* Affectation automatique de presence */
obj->o.gi.gipresence = PRESENT ;

/* Pas de verification possible du stconf */

/* Pas de verification possible du type */
```

```
/* Pas de verification possible de la desc */

/* Pas de verification possible des poss_con et poss_maj */

/* Pas de verification possible des cont_t et cont_c */

/* creation proprement dite */
creat (*obj, code_ret) ;

break ;      /* sortie du case  GR_INFORMATION */


/*************************************************
*                CREATION      INFORMATION       *
*************************************************/

case sinfo :

/* Verifier si l'objet, auquel appartiendra
   l'information , existe */
exist (obj->o.i.icont_t, obj->o.i.icont_c, code_ret) ;
if (*code_ret != OK)
break ;          /* sortie prematuree du case sinfo */

/* Verifier si l'utilisateur a acces a cet objet */
if (obj->o.i.icont_t == sgr_info)
acces_maj  (obj->o.i.icont_t, obj->o.i.icont_c, util.group,
            code_ret) ;
else
acces (obj->o.i.icont_t, obj->o.i.icont_c, util.group,
       code_ret) ;
if (*code_ret != OK)
break ;          /* sortie prematuree du case sinfo */

/* Pas de verification possible du nom */

/* Generation automatique du code */
gen_compt ( &(obj->o.i.icode), code_ret) ;
if (*code_ret != OK)
break ;      /* sortie prematuree du case sinfo */

/* Pas de verification possible de la nature */

/* Affectation de la date du jour --> crdate */
strcpy (obj->o.i.icrdate,date_jour) ;

/* Affectation du code utilisateur --> creat */
obj->o.i.icreat = util.cod ;

/* Affectation automatique de presence */
obj->o.i.ipresence = PRESENT ;

/* Affectation automatique de codate */
strcpy (obj->o.i.icodate,obj->o.i.icrdate) ;

/* Affectation automatique de consult */
```

```
      obj->o.i.iconsult = obj->o.i.icreat ;

      /* Affectation automatique de modate */
      strcpy (obj->o.i.imodate,obj->o.i.icrdate) ;

      /* Affectation automatique de modificat */
      obj->o.i.imodificat = obj->o.i.icreat ;

      /* Pas de verification possible de la desc */

      /* Pas de verification possible du stconf */

      /* Pas de verification possible du type */

      /* Pas de verification possible des poss_con et poss_maj */

      /* Pas de verification possible de exped */

      /* Pas de verification possible des mots_cles */

      /* Pas de verification possible des ref_stck et ref_orig */

      /* Pas de verification possible des cont_t et cont_c */

      /* creation proprement dite */
      creat (*obj, code_ret) ;

      break ;      /* sortie du case INFORMATION */


      /*************************************************
      *           CREATION      INDETERMINEE           *
      *************************************************/

      default  :
      *code_ret= OBJ_INDEF ;

   }  /* sortie du switch obj->type_object */


}     /* FIN DE LA PROCEDURE CREATION */
```

```
# include "suppr.c"


/*****************************************************************
******************************************************************
**                    PROCEDURE     SUPPRESSION                   **
******************************************************************
******************************************************************/

/* Specification : la procedure suppression () procede a la
   suppression d'un objet dans l'E.R. a la demande d'un
   utilisateur. Cette procedure verifie un certain nombre de
   conditions avant de proceder a la suppression proprement dite
   de l'objet.
   A son retour, elle confirme la suppresssion ou renseigne sur
   la cause de la non suppression.
*/


suppression (util, date_jour, obj, code_ret)

/* DECLARATION DES VARIABLES PARAMETRES */

struct  user util ;                 /* donnee */
char    date_jour[26] ;             /* donnee */
tobject *obj ;                      /* donnee et resultat */
int     *code_ret ;                 /* resultat */


/* Remarque : pour *obj, le type et le code de l'objet suffisent,
              le reste est fait par la procedure propr ()    */


{
/* DEBUT DE LA PROCEDURE */

   /* initialisation de la variable *code_ret */
   *code_ret = OK ;

   /* verification du type de l'objet */
   switch (obj->type_object)
   {
     case sutil : break ;
     case sgr_util : break ;
     case senv_rang : break ;
     case spou : break ;
     case sboi : break ;
     case sarm : break ;
     case stir : break ;
     case sfar : break ;
     case sgr_info : break ;
     case sinfo : break ;
     default  : *code_ret = OBJ_INDEF ;
                return (0) ; /* sortie prematuree de suppression*/

   }    /* sortie du switch obj->type_objetct */
```

```
/* lecture des proprietes de l'objet */
propr (obj, code_ret) ;

if (*code_ret != OK)
return (0) ;    /* sortie prematuree de suppression */

/* Traitement du type d'objet particulier */
switch (obj->type_object)
{

  /************************************************
  *              SUPPRESSION   UTILISATEUR        *
  ************************************************/

  case sutil :

  /* pas de suppression possible d'un utilisateur */
  *code_ret = NOT_IMPLEM ;

  break ;      /* sortie du case sutil */


  /************************************************
  *             SUPPRESSION     GROUP_UTIL        *
  ************************************************/

  case sgr_util :

  /* pas de suppression possible d'un groupe d'utilisateurs */
  *code_ret = NOT_IMPLEM ;

  break ;      /* sortie du case sgr_util */


  /************************************************
  *              SUPPRESSION  ENV_RANG            *
  ************************************************/

  case senv_rang :

  /* pas de suppression possible de l'ENV_RANG */
  *code_ret = NOT_IMPLEM ;

  break ;      /* sortie du case senv_rang */


  /************************************************
  *             SUPPRESSION     POUBELLE          *
  ************************************************/

  case spou :

  /* pas de suppression possible de la POUBELLE */
  *code_ret = NOT_IMPLEM ;

  break ;      /* sortie du case spou */
```

```
/************************************************
*            SUPPRESSION     BOITE              *
************************************************/

case sboi :

/* Verifier que l'utilisateur appelant est l'adiminstrateur*/
if (util.cod != cadm)
{  *code_ret = AC_ADM ;
   break ;
}

/* Verifier que la boite est vide */
if (conten (*obj, code_ret) != 0)
{  *code_ret = OBJ_NON_VIDE ;
   break ;       /* sortie prematuree du case sboi */
}

/* Suppression proprement dite */
suppr (obj->type_object, obj->o.b.bcode, code_ret) ;

break ;      /* sortie du case sboi */


/************************************************
*            SUPPRESSION     ARMOIRE            *
************************************************/

case sarm :

/* Verifier que l'utilisateur appelant est l'adiminstrateur*/
if (util.cod != cadm)
{  *code_ret = AC_ADM ;
   break ;
}

/* Verifier que l'armoire est vide */
if (conten (*obj, code_ret) != 0)
{  *code_ret = OBJ_NON_VIDE ;
   break ;       /* sortie prematuree du case sarm */
}

/* Suppression proprement dite */
suppr (obj->type_object, obj->o.a.acode, code_ret) ;

break ;          /* sortie du case sarm */


/************************************************
*               SUPPRESSION     TIROIR           *
************************************************/

case stir :

/* Verifier que l'utilisateur appelant est l'adiminstrateur*/
if (util.cod != cadm)
```

```
{  *code_ret = AC_ADM ;
   break ;
}

/* Verifier que le tiroir est vide */
if (conten (*obj, code_ret) != 0)
{  *code_ret = OBJ_NON_VIDE ;
   break ;       /* sortie prematuree du case stir */
}

/* Suppression proprement dite */
suppr (obj->type_object, obj->o.t.tcode, code_ret) ;

break ;      /* sortie du case stir */


/*************************************************
*                SUPPRESSION     FARDE            *
*************************************************/

case sfar :

/* Verifier que l'utilisateur appelant est l'administrateur
   ou le proprietaire */
if ( (util.cod != cadm) && (util.cod != obj->o.f.fcreat) )
{  *code_ret = AC_SUPP ;
   break ;
}

/* Verifier que la farde est vide */
if (conten (*obj, code_ret) != 0)
{  *code_ret = OBJ_NON_VIDE ;
   break ;       /* sortie prematuree du case sfar */
}

/* Suppression proprement dite */
suppr (obj->type_object, obj->o.f.fcode, code_ret) ;

break;       /* sortie du case sfar */


/*************************************************
*       SUPPRESSION   GROUPE_INFORMATION         *
*************************************************/

case sgr_info :

/* Verifier que l'utilisateur appelant est l'administrateur
   ou le proprietaire */
if ( (util.cod != cadm) && (util.cod != obj->o.gi.gicreat) )
{  *code_ret = AC_SUPP ;
   break ;
}

/* Verifier que le groupe d'informations est vide */
if (conten (*obj, code_ret) != 0)
{  *code_ret = OBJ_NON_VIDE ;
```

```
            break ;        /* sortie prematuree du case sgr_info */
         }

         /* Suppression proprement dite */
         suppr (obj->type_object, obj->o.gi.gicode, code_ret) ;

         break ;       /* sortie du case sgr_info */


         /************************************************
         *                SUPPRESSION     INFORMATION     *
         ************************************************/

         case sinfo :

         /* Verifier que l'utilisateur appelant est l'administrateur
            ou le proprietaire */
         if ( (util.cod != cadm) && (util.cod != obj->o.i.icreat) )
         {  *code_ret = AC_SUPP ;
            break ;
         }

         /* Verifier que l'information est presente */
         if (obj->o.i.ipresence != PRESENT)
         {  *code_ret = INF_NON_PRES ;
            break ;
         }

         /* Suppression proprement dite */
         suppr (obj->type_object, obj->o.i.icode, code_ret) ;

         break ;       /* sortie du case sinfo */

      }        /* sortie du switch obj->type_object */

   }           /* FIN DE LA PROCEDURE SUPPRESSION */
```

```
# include "modif.c"


/*****************************************************************
*****************************************************************
**                    PROCEDURE    MODIFICATION                  **
*****************************************************************
*****************************************************************/

/* Specification : la procedure modification () procede a la
   modification d'une ou plusieurs proprietes d'un objet a la
   demande d'un utilisateur.
   Cette procedure verifie un certain nombre de conditions avant
   de proceder a la modification proprement dite.
   A son retour, elle confirme la modification ou renseigne sur
   la cause de la non possibilite de modifier.
*/


modification (util, date_jour, obj, code_ret)

/* DECLARATION DES VARIABLES PARAMETRES */

struct  user util ;                 /* donnee */
char    date_jour[26] ;             /* donnee */
tobject *obj ;                      /* donnee et resultat */
int     *code_ret ;                 /* resultat */


/* REMARQUE : pour *obj, le type et le code de l'objet suffisent,
              le reste est fait par propr ()    */


{
/* DECLARATION DES VARIABLES LOCALES A LA PROCEDURE */
   tobject cobj ;                        /* variable pour propr () */

/* DEBUT DE LA PROCEDURE */

   /* initialisation de la variable *code_ret */
   *code_ret = OK ;

   /* verification du type de l'objet */
   switch (obj->type_object)
   {
     case sutil : cobj.o.u.ucode = obj->o.u.ucode ; break ;
     case sgr_util : cobj.o.gu.gucode = obj->o.gu.gucode ; break ;
     case senv_rang : cobj.o.e.ecode = obj->o.e.ecode ; break ;
     case spou : cobj.o.p.pcode = obj->o.p.pcode ; break ;
     case sboi : cobj.o.b.bcode = obj->o.b.bcode ; break ;
     case sarm : cobj.o.a.acode = obj->o.a.acode ; break ;
     case stir : cobj.o.t.tcode = obj->o.t.tcode ; break ;
     case sfar : cobj.o.f.fcode = obj->o.f.fcode ; break ;
     case sgr_info : cobj.o.gi.gicode = obj->o.gi.gicode ; break ;
     case sinfo : cobj.o.i.icode = obj->o.i.icode ; break ;
     default  : *code_ret = OBJ_INDEF ;
                return (0) ;
```

```
                /* sortie prematuree de modification */
}     /* sortie du switch obj->type_objetct */

/* lecture des proprietes de l'objet */
cobj.type_object =  obj->type_object ;
propr (&cobj, code_ret) ;

if (*code_ret != OK)
return (0) ;    /* sortie prematuree de modification */

/* Verification de la permission de l'utilisateur */
if (util.per != ECRITURE)
{
*code_ret = AC_ECRIT ;
return (0) ;    /* sortie prematuree de modification */
}

/* Traitement du type d'objet particulier */
switch (obj->type_object)
{

  /************************************************
  *           MODIFICATION   UTILISATEUR          *
  ************************************************/

  case sutil :

  /* Verifier que l'utilisateur appelant est l'adiminstrateur*/
  if (util.cod != cadm)
  {  *code_ret = AC_ADM ;
     break ;
  }

  /* Pas de possibilite de modifier la crdate */
  strcpy (obj->o.u.ucrdate, cobj.o.u.ucrdate) ;

  /* Pas de possibilite de modifier le creat */
  obj->o.u.ucreat = cobj.o.u.ucreat ;

  /* Modification proprement dite */
  modif (*obj, code_ret) ;

  break ;      /* sortie du case sutil */


  /************************************************
  *           MODIFICATION     GROUP_UTIL          *
  ************************************************/

  case sgr_util :

  /* Verifier que l'utilisateur appelant est l'adiminstrateur*/
  if (util.cod != cadm)
  {  *code_ret = AC_ADM ;
     break ;
  }
```

```
/* Pas de possibilite de modifier le nom */
strcpy (obj->o.gu.gunom, cobj.o.gu.gunom) ;

/* Pas de possibilite de modifier la crdate */
strcpy (obj->o.gu.gucrdate, cobj.o.gu.gucrdate) ;

/* Pas de possibilite de modifier le creat */
obj->o.gu.gucreat = cobj.o.gu.gucreat ;

/* Modification proprement dite */
modif (*obj, code_ret) ;

break ;      /* sortie du case sgr_util */


/*************************************************
*              MODIFICATION  ENV_RANG             *
*************************************************/

case senv_rang :

/* Verifier que l'utilisateur appelant est l'adiminstrateur*/
if (util.cod != cadm)
{  *code_ret = AC_ADM ;
   break ;
}

/* Pas de possibilite de modifier la crdate */
strcpy (obj->o.e.ecrdate, cobj.o.e.ecrdate) ;

/* Pas de possibilite de modifier le creat */
obj->o.e.ecreat = cobj.o.e.ecreat ;

/* Modification proprement dite */
modif (*obj, code_ret) ;

break ;      /* sortie du case senv_rang */


/*************************************************
*           MODIFICATION     POUBELLE            *
*************************************************/

case spou :

/* Verifier que l'utilisateur appelant est l'adiminstrateur*/
if (util.cod != cadm)
{  *code_ret = AC_ADM ;
   break ;
}

/* Pas de possibilite de modifier la crdate */
strcpy (obj->o.p.pcrdate, cobj.o.p.pcrdate) ;

/* Pas de possibilite de modifier le creat */
obj->o.p.pcreat = cobj.o.p.pcreat ;
```

```
/* Modification proprement dite */
modif (*obj, code_ret) ;

break ;      /* sortie du case spou */


/*************************************************
*            MODIFICATION     BOITE              *
*************************************************/

case sboi :

/* Verifier que l'utilisateur appelant est l'adiminstrateur*/
if (util.cod != cadm)
{  *code_ret = AC_ADM ;
   break ;
}

/* Pas de possibilite de modifier la crdate */
strcpy (obj->o.b.bcrdate, cobj.o.b.bcrdate) ;

/* Pas de possibilite de modifier le creat */
obj->o.b.bcreat = cobj.o.b.bcreat ;

/* Modification proprement dite */
modif (*obj, code_ret) ;

break ;      /* sortie du case sboi */


/*************************************************
*             MODIFICATION     ARMOIRE           *
*************************************************/

case sarm :

/* Verifier que l'utilisateur appelant est l'adiminstrateur*/
if (util.cod != cadm)
{  *code_ret = AC_ADM ;
   break ;
}

/* Pas de possibilite de modifier la crdate */
strcpy (obj->o.a.acrdate, cobj.o.a.acrdate) ;

/* Pas de possibilite de modifier le creat */
obj->o.a.acreat = cobj.o.a.acreat ;

/* Modification proprement dite */
modif (*obj, code_ret) ;

break ;          /* sortie du case sarm */
```

```
/************************************************
*              MODIFICATION     TIROIR          *
************************************************/

case stir :

/* Verifier que l'utilisateur appelant est l'adiminstrateur*/
if (util.cod != cadm)
{  *code_ret = AC_ADM ;
   break ;
}

/* Pas de possibilite de modifier la crdate */
strcpy (obj->o.t.tcrdate, cobj.o.t.tcrdate) ;

/* Pas de possibilite de modifier le creat */
obj->o.t.tcreat = cobj.o.t.tcreat ;

/* Modification proprement dite */
modif (*obj, code_ret) ;

break ;      /* sortie du case stir */


/************************************************
*              MODIFICATION     FARDE           *
************************************************/

case sfar :

/* Verifier que l'utilisateur appelant est l'administrateur
   ou le proprietaire */
if ( (util.cod != cadm) && (util.cod != obj->o.f.fcreat) )
{  *code_ret = AC_MODIF ;
   break ;
}

/* Pas de possibilite de modifier la crdate */
strcpy (obj->o.f.fcrdate, cobj.o.f.fcrdate) ;

/* Pas de possibilite de modifier le creat */
obj->o.f.fcreat = cobj.o.f.fcreat ;

/* Modification proprement dite */
modif (*obj, code_ret) ;

break;       /* sortie du case sfar */
```

```
/*************************************************
*      MODIFICATION    GROUPE_INFORMATION       *
*************************************************/

case sgr_info :

/* Verifier que l'utilisateur appelant est l'administrateur
   ou le proprietaire */
if ( (util.cod != cadm) && (util.cod != obj->o.gi.gicreat) )
{  *code_ret = AC_MODIF ;
   break ;
}

/* Pas de possibilite de modifier la presence */
obj->o.gi.gipresence = cobj.o.gi.gipresence ;

/* Pas de possibilite de modifier la crdate */
strcpy (obj->o.gi.gicrdate, cobj.o.gi.gicrdate) ;

/* Pas de possibilite de modifier le creat */
obj->o.gi.gicreat = cobj.o.gi.gicreat ;

/* Modification proprement dite */
modif (*obj, code_ret) ;

break ;      /* sortie du case sgr_info */


/*************************************************
*           MODIFICATION     INFORMATION         *
*************************************************/

case sinfo :

/* Verifier que l'utilisateur appelant est l'administrateur
   ou le proprietaire */
if ( (util.cod != cadm) && (util.cod != obj->o.i.icreat) )
{  *code_ret = AC_MODIF ;
   break ;
}

/* Pas de possibilite de modifier la presence */
obj->o.i.ipresence = cobj.o.i.ipresence ;

/* Pas de possibilite de modifier la crdate */
strcpy (obj->o.i.icrdate, cobj.o.i.icrdate) ;

/* Pas de possibilite de modifier le creat */
obj->o.i.icreat = cobj.o.i.icreat ;

/* Pas de possibilite de modifier la codate */
strcpy (obj->o.i.icodate, cobj.o.i.icodate) ;

/* Pas de possibilite de modifier le consult */
obj->o.i.iconsult = cobj.o.i.iconsult ;
```

```
        /* Pas de possibilite de modifier la modate */
        strcpy (obj->o.i.imodate, cobj.o.i.imodate) ;

        /* Pas de possibilite de modifier le modificat */
        obj->o.i.imodificat = cobj.o.i.imodificat ;

        /* Modification proprement dite */
        modif (*obj, code_ret) ;

        break ;      /* sortie du case sinfo */

    }       /* sortie du switch obj->type_object */

}           /* FIN DE LA PROCEDURE MODIFICATION */
```

```
# include "consul.c"


/*****************************************************************
*****************************************************************
**                    PROCEDURE    CONSULTATION                  **
*****************************************************************
*****************************************************************/

/* Specification : la procedure concultation () procede a la
   consultation d'une information ou d'un groupe d'informations
   a la demande d'un utilisateur.
   Cette procedure verifie un certain nombre de conditions avant
   de proceder a la consultation proprement dite.
   A son retour, elle confirme la consultation ou renseigne sur
   la cause de la non possibilite de consulter.
*/


consultation (util, date_jour, obj, mode, code_ret)

/* DECLARATION DES VARIABLES PARAMETRES */

struct  user util ;                     /* donnee */
char    date_jour[26] ;                 /* donnee */
tobject *obj ;                          /* donnee et resultat */
char    mode ;                          /* donnee */
int     *code_ret ;                     /* resultat */


/* Remarque : pour le parametre *obj, seul de type et le code
   de l'objet suffisent pour l'ouverture. Pour la fermeture,
   il faut en egalement la date du jour et l'util */


{
/* DEBUT DE LA PROCEDURE */

   /* initialisation de la variable *code_ret */
   *code_ret = OK ;

   /* Verification du mode de consultation */
   switch (mode)
   {
   case OUVERTURE : break ;
   case FERMETURE : break ;
   default : *code_ret = MODE_INEXIST ;
             return (0) ;  /* sortie prematuree de consulation */
   }    /* sortie du case mode */

   /* Traitement du type d'objet particulier */
   switch (obj->type_object)
   {
```

```
/************************************************
*       CONSULTATION   GROUPE_INFORMATION      *
************************************************/

case sgr_info :

/* Pas de verification necessaire de la permission de
   l'utilisateur, LECTURE suffit pour la consultation */

/* lecture des proprietes du groupe d'informations */
propr (obj, code_ret) ;

if (*code_ret != OK)
return (0) ;        /* sortie prematuree de consultation */

/* Verification de la confidentialite du groupe
   d'informations */
if ( (obj->o.gi.gistconf[0] == CONFIDENTIEL)
     && (obj->o.gi.gicreat != util.cod) )
{
*code_ret = AC_CREAT ;
return (0) ;    /* sortie prematuree de consultation */
}
else
/* Verification de la possibilite d'acces au groupe
   d'informations */
{
   if ( (str_in (util.group, obj->o.gi.giposs_maj) == 0)
        && (str_in (util.group, obj->o.gi.giposs_con) == 0) )
   {
   *code_ret = AC_CONSUL ;
   return (0) ;  /* sortie prematuree de consultation */
   }
}

/* Traitement en fonction du mode de consultation */
switch (mode)
{
case OUVERTURE :
    /* Verification de la presence */
    if (obj->o.gi.gipresence == MAJ)
    {   *code_ret = DISP_OBJ ;
        return (0) ;     /* si en mise a jour */
    }

     /* Ajoute d'une consultation
        dans obj->o.gi.gipresence si possible
        MAXPRES - 1 est la derniere consultation possible ;
        MAXPRES represente la mise a jour (MAJ) */

     if ( obj->o.gi.gipresence < (MAXPRES - 2) )
     obj->o.gi.gipresence++ ;
     else
     { *code_ret = ERR_PRES ;
       return (0) ;  /* sortie prematuree de consultation */
     }
```

```
    break ;   /* sortie case OUVERTURE */

case FERMETURE :
     /* Suppression d'une consultation
        dans obj->o.gi.gipresence
        si possible  */
     if ( (obj->o.gi.gipresence - 1) < 0 )
     { *code_ret = CONT_PRES ;
       return (0) ;  /* sortie prematuree de consultation */
     }
     else
     obj->o.gi.gipresence-- ;

}    /* sortie du switch mode */

 /* consultation proprement dite */
 consul (obj, mode ,code_ret) ;

 break ;      /* sortie du case sgr_info  */


 /*************************************************
 *           CONSULTATION     INFORMATION         *
 *************************************************/

 case sinfo :

 /* Pas de verification necessaire de la permission de
    l'utilisateur, LECTURE suffit pour la consultation */

 /* lecture des proprietes de l'information */
 propr (obj, code_ret) ;

 if (*code_ret != OK)
 return (0) ;        /* sortie prematuree de consultation */

 /* Verification de la confidentialite de l'information */
 if ( (obj->o.i.istconf[0] == CONFIDENTIEL)
      && (obj->o.i.icreat != util.cod) )
 {
 *code_ret = AC_CREAT ;
 return (0) ;   /* sortie prematuree de consultation */
 }
 else
 /* Verification de la possibilite d'acces a  l'information */
 {
    if ( (str_in (util.group, obj->o.i.iposs_maj) == 0)
         && (str_in (util.group, obj->o.i.iposs_con) == 0) )
    {
    *code_ret = AC_CONSUL ;
    return (0) ;  /* sortie prematuree de consultation */
    }
 }

/* Traitement en fonction du mode de consultation */
switch (mode)
```

```
{
case OUVERTURE :
     /* Verification de la presence */
     if (obj->o.i.ipresence == MAJ)
     {    *code_ret = DISP_OBJ ;
          return (0) ;     /* si en mise a jour */
     }

      /* Ajoute d'une consultation
         dans obj->o.i.ipresence si possible
         MAXPRES - 1 est la derniere consultation possible ;
         MAXPRES represente la mise a jour (MAJ) */

      if ( obj->o.i.ipresence < (MAXPRES - 2) )
      obj->o.i.ipresence++ ;
      else
      { *code_ret = ERR_PRES ;
        return (0) ;  /* sortie prematuree de consultation */
      }
      break ;  /* sortie du case OUVERTURE */

case FERMETURE :
      /* Suppression d'une consultation dans obj->o.i.ipresence
         si possible  */
      if ( (obj->o.i.ipresence -1) < 0 )
      { *code_ret = CONT_PRES ;
        return (0) ;  /* sortie prematuree de consultation */
      }
      else
      obj->o.i.ipresence-- ;

}    /* sortie du switch mode */

  /* Affectation de la date du jour a la date de modification*/
  strcpy (obj->o.i.icodate,date_jour) ;

  /* Affectation de l'utilisateur au modificateur */
  obj->o.i.iconsult = util.cod ;

  /* consultation proprement dite */
  consul (obj, mode, code_ret) ;

  break ;      /* sortie du case sinfo */


  /*************************************************
  *          CONSULTATION     INDETERMINEE         *
  *************************************************/

  default  :
  *code_ret= OBJ_INDEF ;

}      /* sortie du switch obj->type_object */

} ;       /* FIN DE LA PROCEDURE CONSULTATION */
```

```
# include "maj.c"


/**************************************************************************
**************************************************************************
**                      PROCEDURE     MISE_A_JOUR                         **
**************************************************************************
**************************************************************************/

/* Specification : la procedure mise_a_jour () procede a la
   mise_a_jour d'une information ou d'un groupe d'informations
   a la demande d'un utilisateur.
   Cette procedure verifie un certain nombre de conditions avant
   de proceder a la mise_a_jour proprement dite.
   A son retour, elle confirme la mise_a_jour ou renseigne sur
   la cause de la non possibilite de mettre a jour.
*/


mise_a_jour (util, date_jour, obj, mode, code_ret)

/* DECLARATION DES VARIABLES PARAMETRES */

struct  user util ;                     /* donnee */
char    date_jour[26] ;                 /* donnee */
tobject *obj ;                          /* donnee et resultat */
char    mode ;                          /* donnee */
int     *code_ret ;                     /* resultat */


/* Remarque : pour le parametre *obj, seul de type et le code
   de l'objet suffisent pour l'ouverture. Pour la fermeture,
   il faut en egalement la date du jour et l'util */


{
/* DEBUT DE LA PROCEDURE */

   /* initialisation de la variable *code_ret */
   *code_ret = OK ;

   /* Verification du mode de mise_a_jour */
   switch (mode)
   {
   case OUVERTURE : break ;
   case FERMETURE : break ;
   default : *code_ret = MODE_INEXIST ;
             return (0) ;  /* sortie prematuree de mise_a_jour */
   }    /* sortie du case mode */

   /* Traitement du type d'objet particulier */
   switch (obj->type_object)
   {
```

```
/************************************************
*        MISE_A_JOUR    GROUPE_INFORMATION      *
************************************************/

case sgr_info :

/* Verification de la permission de l'utilisateur */
if (util.per != ECRITURE)
{
*code_ret = AC_ECRIT ;
return (0) ;        /* sortie prematuree de mise_a_jour */
}

/* lecture des proprietes du groupe d'informations */
propr (obj, code_ret) ;

if (*code_ret != OK)
return (0) ;        /* sortie prematuree de mise_a_jour */

/* Verification de la confidentialite du groupe
   d'informations */
if ( (obj->o.gi.gistconf[0] == CONFIDENTIEL)
     && (obj->o.gi.gicreat != util.cod) )
{
*code_ret = AC_CREAT ;
return (0) ;    /* sortie prematuree de mise_a_jour */
}
else
/* Verification de la possibilite d'acces au groupe
   d'informations */
{
   if (str_in (util.group, obj->o.gi.giposs_maj) == 0)
   {
   *code_ret = AC_MAJ ;
   return (0) ;  /* sortie prematuree de mise_a_jour */
   }
}

/* Traitement en fonction du mode de mise_a_jour */
switch (mode)
{
case OUVERTURE :
     /* Verification de la presence */
     if (obj->o.gi.gipresence != PRESENT)
     {   *code_ret = DISP_OBJ ;
         return (0) ; /* si pas PRESENT, abandon traitement */
     }

     /* Met mise_a_jour dans obj->o.gi.gipresence */
     obj->o.gi.gipresence = MAJ ;

     break ;  /* sortie case OUVERTURE */

case FERMETURE :
     /* Suppression de la mise_a_jour
        dans obj->o.gi.gipresence
```

```
          si possible  */

        if (obj->o.gi.gipresence != MAJ)
        { *code_ret = CONT_PRES ;
          return (0) ;  /* sortie prematuree de mise_a_jour */
        }
        else
        obj->o.gi.gipresence = PRESENT ;

}     /* sortie du switch mode */

  /* mise_a_jour proprement dite */
  consul (obj, mode ,code_ret) ;

  break ;       /* sortie du case sgr_info  */


  /*************************************************
  *            MISE_A_JOUR     INFORMATION         *
  *************************************************/

  case sinfo :

  /* Verification de la permission de l'utilisateur */
  if (util.per != ECRITURE)
  {
  *code_ret = AC_ECRIT ;
  return (0) ;         /* sortie prematuree de mise_a_jour */
  }

  /* lecture des proprietes de l'information */
  propr (obj, code_ret) ;

  if (*code_ret != OK)
  return (0) ;         /* sortie prematuree de mise_a_jour */

  /* Verification de la confidentialite de l'information */
  if ( (obj->o.i.istconf[0] == CONFIDENTIEL)
       && (obj->o.i.icreat != util.cod) )
  {
  *code_ret = AC_CREAT ;
  return (0) ;    /* sortie prematuree de mise_a_jour */
  }
  else
  /* Verification de la possibilite d'acces a  l'information */
  {
     if (str_in (util.group, obj->o.i.iposs_maj) == 0)
     {
     *code_ret = AC_MAJ ;
     return (0) ;  /* sortie prematuree de mise_a_jour */
     }
  }

 /* Traitement en fonction du mode de mise_a_jour */
 switch (mode)
 {
 case OUVERTURE :
```

```
        /* Verification de la presence */
        if (obj->o.i.ipresence != PRESENT)
        {    *code_ret = DISP_OBJ ;
             return (0) ; /* si pas present, abandon traitement */
        }

        /* Met mise_a_jour dans obj->o.i.ipresence */
         obj->o.i.ipresence = MAJ ;

        break ;  /* sortie case OUVERTURE */

    case FERMETURE :
        /* Suppression d'une mise_a_jour dans obj->o.i.ipresence
           si possible  */
        if (obj->o.i.ipresence != MAJ)
        { *code_ret = CONT_PRES ;
          return (0) ;  /* sortie prematuree de mise_a_jour */
        }
        else
        obj->o.i.ipresence = PRESENT ;

    }    /* sortie du case mode */

     /* Affectation de la date du jour a la date de modification*/
     strcpy (obj->o.i.icodate,date_jour) ;

     /* Affectation de l'utilisateur au modificateur */
     obj->o.i.iconsult = util.cod ;

     /* mise_a_jour proprement dite */
     consul (obj, mode, code_ret) ;

     break ;      /* sortie du case sinfo */


     /************************************************
     *          MISE_A_JOUR      INDETERMINEE          *
     ************************************************/

     default  :
     *code_ret= OBJ_INDEF ;

   }      /* sortie du switch obj->type_object */

} ;       /* FIN DE LA PROCEDURE MISE_A_JOUR */
```

```
#  include "rech.c"


/*****************************************************************
*****************************************************************
**                    PROCEDURE     RECHERCHE                    **
*****************************************************************
*****************************************************************/

/* Specification : la procedure recherche () procede a la
   recherche de tous les objets d'un certain type repondant
   a certaines conditions donnees par l'utilisateur.
   A son retour, elle renvoie le nombre d'objets trouves
   ainsi que la liste des codes de ceux-ci.
*/


recherche (util, date_jour, obj, cond, res, code_ret)

/* DECLARATION DES VARIABLES PARAMETRES */

struct  user  util ;                    /* donnee */
char    date_jour[26] ;                 /* donnee */
tobject *obj ;                          /* donnee et resultat */
char    cond[] ;                        /* donnee */
list    *res ;                          /* resulat */
int     *code_ret ;                     /* resultat */

{
/* DEBUT DE LA PROCEDURE */

   /* initialisation de la variable *code_ret */
   *code_ret = OK ;

   /* Traitement du type d'objet particulier */
   switch (obj->type_object)
   {

     /*************************************************
     *                RECHERCHE UTILISATEUR           *
     *************************************************/

     case sutil :

     /* Aucune verification ne doit etre faite a ce niveau pour
        l'appel de la fonction rech ()  */

     /* recherche proprement dite */
     rech (obj->type_object, cond, res, code_ret) ;
     /* code_ret est une adresse & */

     break ;      /* sortie du case sutil */
```

```
/************************************************
*            RECHERCHE       GR_UTIL            *
************************************************/

case sgr_util :

/* Aucune verification ne doit etre faite a ce niveau pour
   l'appel de la fonction rech ()  */

/* recherche proprement dite */
rech (obj->type_object, cond, res, code_ret) ;

break ;      /* sortie du case sgr_util */


/************************************************
*               RECHERCHE   ENV_RANG             *
************************************************/

case senv_rang :

/* Aucune verification ne doit etre faite a ce niveau pour
   l'appel de la fonction rech ()  */

/* recherche proprement dite */
rech (obj->type_object, cond, res, code_ret) ;

break ;      /* sortie du case senv_rang */


/************************************************
*               RECHERCHE     POUBELLE           *
************************************************/

case spou :

/* Aucune verification ne doit etre faite a ce niveau pour
   l'appel de la fonction rech ()  */

/* recherche proprement dite */
rech (obj->type_object, cond, res, code_ret) ;

break ;      /* sortie du case spou */


/************************************************
*               RECHERCHE     BOITE              *
************************************************/

case sboi :

/* Aucune verification ne doit etre faite a ce niveau pour
   l'appel de la fonction rech ()  */

/* recherche proprement dite */
rech (obj->type_object, cond, res, code_ret) ;
```

```
break ;      /* sortie du case sboi */


/************************************************
*                RECHERCHE     ARMOIRE           *
************************************************/

case sarm :

/* Aucune verification ne doit etre faite a ce niveau pour
   l'appel de la fonction rech ()  */

/* recherche proprement dite */
rech (obj->type_object, cond, res, code_ret) ;

break ;          /* sortie du case sarm */


/************************************************
*                RECHERCHE     TIROIR            *
************************************************/

case stir :

/* Aucune verification ne doit etre faite a ce niveau pour
   l'appel de la fonction rech ()  */

/* recherche proprement dite */
rech (obj->type_object, cond, res, code_ret) ;

break ;      /* sortie du case TIROIR */


/************************************************
*                RECHERCHE     FARDE             *
************************************************/

case sfar :

/* Aucune verification ne doit etre faite a ce niveau pour
   l'appel de la fonction rech ()  */

/* recherche proprement dite */
rech (obj->type_object, cond, res, code_ret) ;

break;       /* sortie du case FARDE */


/************************************************
*        RECHERCHE     GROUPE_INFORMATION       *
************************************************/

case sgr_info :

/* Aucune verification ne doit etre faite a ce niveau pour
   l'appel de la fonction rech ()  */
```

```
        /* recherche proprement dite */
        rech (obj->type_object, cond, res, code_ret) ;

        break ;      /* sortie du case  GR_INFORMATION */


        /************************************************
        *                RECHERCHE     INFORMATION      *
        ************************************************/

        case sinfo :

        /* Aucune verification ne doit etre faite a ce niveau pour
           l'appel de la fonction rech ()  */

        /* recherche proprement dite */
        rech (obj->type_object, cond, res, code_ret) ;

        break ;      /* sortie du case INFORMATION */


        /************************************************
        *          RECHERCHE     INDETERMINEE           *
        ************************************************/

        default  :
        *code_ret= OBJ_INDEF ;

    }  /* sortie du switch obj->type_object */


}      /* FIN DE LA PROCEDURE RECHERCHE */
```

```
#  include "stdio.h"
#  include "lib_c.c"
#  include "lib_v.c"

/*****************************************************************
******************************************************************
**                    PROCEDURE     PROPRIETE                    **
******************************************************************
******************************************************************/

/* Specification : la procedure propriete () recherche toutes
   les proprietes liees a un objet a la demande d'un utilisateur.
   A son retour, elle renvoie ces proprietes ou renseigne sur
   la cause de son echec.
*/


propriete (util, date_jour, obj, code_ret)

/* DECLARATION DES VARIABLES PARAMETRES */

struct  user  util ;                    /* donnee */
char    date_jour[26] ;                 /* donnee */
tobject *obj ;                          /* donnee et resultat */
int     *code_ret ;                     /* resultat */

{
/* DEBUT DE LA PROCEDURE */

   /* initialisation de la variable *code_ret */
   *code_ret = OK ;

   /* Traitement du type d'objet particulier */
   switch (obj->type_object)
   {

     /*************************************************
     *                PROPRIETE UTILISATEUR           *
     *************************************************/

     case sutil :

     /* Aucune verification ne doit etre faite a ce niveau pour
        l'appel de la fonction propr ()  */

     /* propriete proprement dite */
     propr (obj, code_ret) ;     /* code_ret est une adresse & */

     break ;      /* sortie du case sutil */


     /*************************************************
     *            PROPRIETE       GR_UTIL             *
     *************************************************/

     case sgr_util :
```

```
/* Aucune verification ne doit etre faite a ce niveau pour
   l'appel de la fonction propr ()  */

/* propriete proprement dite */
propr (obj, code_ret) ;

break ;      /* sortie du case sgr_util */


/************************************************
*               PROPRIETE   ENV_RANG            *
************************************************/

case senv_rang :

/* Aucune verification ne doit etre faite a ce niveau pour
   l'appel de la fonction propr ()  */

/* propriete proprement dite */
propr (obj, code_ret) ;

break ;      /* sortie du case senv_rang */


/************************************************
*               PROPRIETE     POUBELLE          *
************************************************/

case spou :

/* Aucune verification ne doit etre faite a ce niveau pour
   l'appel de la fonction propr ()  */

/* propriete proprement dite */
propr (obj, code_ret) ;

break ;      /* sortie du case spou */


/************************************************
*               PROPRIETE     BOITE             *
************************************************/

case sboi :

/* Aucune verification ne doit etre faite a ce niveau pour
   l'appel de la fonction propr ()  */

/* propriete proprement dite */
propr (obj, code_ret) ;

break ;      /* sortie du case sboi */
```

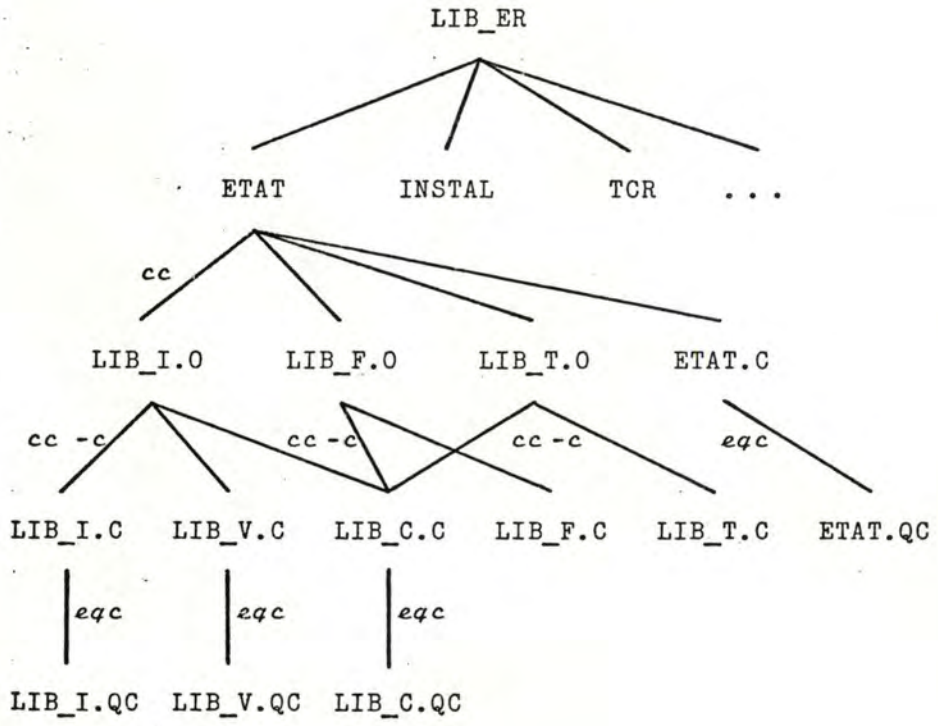
```
/************************************************
*                PROPRIETE     ARMOIRE            *
************************************************/

case sarm :

/* Aucune verification ne doit etre faite a ce niveau pour
   l'appel de la fonction propr ()  */

/* propriete proprement dite */
propr (obj, code_ret) ;

break ;          /* sortie du case sarm */


/************************************************
*                PROPRIETE     TIROIR             *
************************************************/

case stir :

/* Aucune verification ne doit etre faite a ce niveau pour
   l'appel de la fonction propr ()  */

/* propriete proprement dite */
propr (obj, code_ret) ;

break ;      /* sortie du case TIROIR */


/************************************************
*                PROPRIETE     FARDE              *
************************************************/

case sfar :

/* Aucune verification ne doit etre faite a ce niveau pour
   l'appel de la fonction propr ()  */

/* propriete proprement dite */
propr (obj, code_ret) ;

break;       /* sortie du case FARDE */


/************************************************
*        PROPRIETE     GROUPE_INFORMATION       *
************************************************/

case sgr_info :

/* Aucune verification ne doit etre faite a ce niveau pour
   l'appel de la fonction propr ()  */

/* propriete proprement dite */
propr (obj, code_ret) ;
```

```
      break ;      /* sortie du case  GR_INFORMATION */

      /************************************************
      *                PROPRIETE     INFORMATION       *
      ************************************************/

      case sinfo :

      /* Aucune verification ne doit etre faite a ce niveau pour
         l'appel de la fonction propr ()  */

      /* propriete proprement dite */
      propr (obj, code_ret) ;

      break ;      /* sortie du case INFORMATION */


      /************************************************
      *          PROPRIETE     INDETERMINEE           *
      ************************************************/

      default  :
      *code_ret= OBJ_INDEF ;

   }  /* sortie du switch obj->type_object */


}     /* FIN DE LA PROCEDURE PROPRIETE */
```

ANNEXE B : SOURCES DES PROGRAMMES DE TEST.

L'annexe B est composée de 2 parties : la première reprend le fichier MAKEFILE et la seconde reprend l'ensemble des programmes de test ainsi que leurs résultats.

L'utilitaire MAKE du système d'exploitation UNIX a permis de faciliter le développement de notre application opération par opération. Cet utilitaire a pour entrée un fichier intitulé MAKEFILE où sont décrits tous les fichiers ainsi que les actions à activer pour mettre à jour la dernière version de l'application. Il utilise à cet effet la date de dernière mise à jour des fichiers concernés. On peut encore comparer la structure du fichier MAKEFILE à une structure en réseau représentant les dépendances entre fichiers. Cette structure pourrait se schématiser de la manière suivante :



Les autres fichiers de cette annexe contiennent les tests qui ont été effectués sur les opérations de l'annexe A.

La convention adoptée pour la notation des fichiers est la même que celle de l'annexe A avec l'ajout de l'extension 'OUT' qui correspondra au résultat apparaissant à l'écran lors de l'exécution du programme de test correspondant. Tous les tests ont été réalisés à partir d'une configuration de l'E.R. identique obtenu par l'exécution du programme INSTAL suivi du programme TCR.

L'ensemble de ces programmes n'a pas été repris dans ce volume. Une version complète de l'annexe B est disponible auprès du promoteur du mémoire, Mr. Lesuisse, pour les personnes qui seraient désireuses de la consulter.

```
lib_er : etat instal destal tcr tsup tpr tcoma tmo tconten tre

etat :    lib_i.o lib_f.o lib_t.o etat.c
     cc etat.c  lib_t.o lib_f.o lib__i.o /mnt/ingres/lib/libqlib
        /mnt/ingres/lib/compatlib -lm -lc -o etat

etat.c : lib_c.qc lib_v.qc etat.qc
     eqc etat.qc

tcr :   lib_i.o lib_f.o creation.o tcr.c
     cc tcr.c lib_i.o lib_f.o creation.o ...
        /mnt/ingres/lib/libqlib /mnt/ingres/lib/compatlib ...
        -lm -lc -o tcr

creation.o : lib_i.o lib_f.o creat.c creation.c
     cc -c creation.c lib_f.o lib_i.o /mnt/ingres/lib/libqlib ...
        /mnt/ingres/lib/compatlib -lm -lc

creat.c : lib_c.qc lib_v.qc creat.qc
     eqc creat.qc

tsup : lib_i.o lib_f.o conten.o suppression.o tsup.c
     cc tsup.c suppression.o conten.o lib_f.o lib_i.o ...
        /mnt/ingres/lib/libqlib /mnt/ingres/lib/compatlib ...
        -lm -lc -o tsup

suppression.o : lib_i.o lib_f.o conten.o suppr.c suppression.c
     cc -c suppression.c conten.o lib_f.o lib_i.o ...
        /mnt/ingres/lib/libqlib /mnt/ingres/lib/compatlib -lm -lc

suppr.c : lib_c.qc lib_v.qc suppr.qc
     eqc suppr.qc

tpr : lib_i.o lib_f.o lib_t.o propriete.o tpr.c
     cc tpr.c propriete.o lib_t.o lib_f.o lib_i.o ...
        /mnt/ingres/lib/libqlib /mnt/ingres/lib/compatlib ...
        -lm -lc -o tpr

propriete.o : lib_i.o lib_f.o propriete.c
     cc -c propriete.c lib_f.o lib_i.o /mnt/ingres/lib/libqlib ...
        /mnt/ingres/lib/compatlib -lm -lc

tcoma : lib_i.o lib_f.o consultation.o mise_a_jour.o tcoma.c
     cc tcoma.c mise_a_jour.o consultation.o lib_f.o lib_i.o ...
        /mnt/ingres/lib/libqlib /mnt/ingres/lib/compatlib ...
        -lm -lc -o tcoma

consultation.o : lib_i.o lib_f.o consul.c consultation.c
     cc -c consultation.c lib_f.o lib_i.o /mnt/ingres/lib/libqlib
        /mnt/ingres/lib/compatlib -lm -lc

consul.c : lib_c.qc lib_v.qc consul.qc
     eqc consul.qc

mise_a_jour.o : lib_i.o lib_f.o maj.c mise_a_jour.c
     cc -c mise_a_jour.c lib_f.o lib_i.o /mnt/ingres/lib/libqlib
        /mnt/ingres/lib/compatlib -lm -lc
```

```
maj.c : lib_c.qc lib_v.qc maj.qc
     eqc maj.qc

tmo : lib_i.o lib_f.o modification.o tmo.c
     cc tmo.c modification.o lib_f.o lib_i.o ...
        /mnt/ingres/lib/libqlib /mnt/ingres/lib/compatlib ...
        -lm -lc -o tmo

modification.o : lib_i.o lib_f.o modif.c modification.c
     cc -c modification.c lib_f.o lib_i.o /mnt/ingres/lib/libqlib
        /mnt/ingres/lib/compatlib -lm -lc

modif.c : lib_c.qc lib_v.qc modif.qc
     eqc modif.qc

tre : lib_i.o lib_f.o recherche.o tre.c
     cc tre.c lib_i.o lib_f.o recherche.o ...
        /mnt/ingres/lib/libqlib /mnt/ingres/lib/compatlib ...
        -lm -lc -o tre

recherche.o : lib_i.o lib_f.o rech.c recherche.c
     cc -c recherche.c lib_f.o lib_i.o /mnt/ingres/lib/libqlib
        /mnt/ingres/lib/compatlib -lm -lc

rech.c : lib_c.qc lib_v.qc rech.qc
     eqc rech.qc

tconten : lib_c.c lib_v.c lib_i.o conten.c tconten.c
     cc tconten.c lib_i.o /mnt/ingres/lib/libqlib ...
        /mnt/ingres/lib/compatlib -lm -lc -o tconten

conten.o : lib_c.c lib_v.c conten.c
     cc -c conten.c /mnt/ingres/lib/libqlib ...
        /mnt/ingres/lib/compatlib -lm -lc

conten.c : lib_c.qc lib_v.qc conten.qc
     eqc conten.qc

lib_f.o : lib_c.c lib_f.c
     cc -c lib_f.c /mnt/ingres/lib/libqlib ...
        /mnt/ingres/lib/compatlib -lm -lc

lib_i.o : lib_c.c lib_v.c lib_i.c
     cc -c lib_i.c /mnt/ingres/lib/libqlib ...
        /mnt/ingres/lib/compatlib -lm -lc

lib_i.c : lib_c.qc lib_v.qc lib_i.qc
     eqc lib_i.qc

lib_c.c : lib_c.qc
     eqc lib_c.qc

lib_v.c : lib_v.qc
     eqc lib_v.qc
```

```
lib_t.o : lib_c.c lib_t.c
     cc -c lib_t.c /mnt/ingres/lib/libqlib ...
        /mnt/ingres/lib/compatlib -lm -lc

destal : destal.c
     cc destal.c -o destal

instal : lib_c.c lib_v.qc instal.c
     cc instal.c /mnt/ingres/lib/libqlib ...
        /mnt/ingres/lib/compatlib -lm -lc -o instal

instal.c : lib_c.qc lib_v.qc instal.qc
     eqc instal.qc
```

ANNEXE C : SOURCES DE FONCTIONS LISP DE GENERATION AUTOMATIQUE DE REQUETES INGRES.

La présente annexe reprend les sources complètes des fonctions LISP qui ont été utilisées à titre illustratif au chapitre 5.

Les fichiers correspondant accompagnés d'une note explicative de leur contenu sont :

START.LSP : définition de la fonction 'SUBST' qui permet de remplacer une occurrence d'atome ou de liste par un(e) autre à l'intérieur d'un atome ou d'une liste.

FORM1.LSP : définition de la fonction de formatage 'FORMATE' permettant de sortir à l'écran la requête Ingres générée.

FORM2.LSP : définition de la même fonction de formatage 'FORMATE' mais permettant de sortir sur le fichier 'GENERE.DOC' la requête Ingres générée.

GENERE.LSP : définition de la fonction pricipale 'GENERE' utilisant notamment les fonctions 'SUBST' et 'FORMATE' et qui est appelée pour générer une requête Ingres.

FICHIER START.LSP :

```
(DEFUN SUBST
  (LAMBDA (A B C)
      (COND ((NULL C) C)
         ((ATOM C) (COND ((EQUAL A C) B)
                         (T C)))
         (T (CONS (SUBST A B (CAR C))
                  (SUBST A B (CDR C)))))))
```

FICHIER FORM1.LSP :

```
(DEFUN FORMATE
   (LAMBDA (X)
      (PRINTC (CONCAT (CAAR X) '" (" (CADAR X) '=
              (CAR (CDDAR X)) '. (CADR (CDDAR X)) ') ) )
      (PRINTC (CONCAT (CAADR X) '" " (CAR (CDADR X)) '.
                      (CADR (CDADR X)) (CADDR (CDADR X)) ))
      (PRIN (CAR (CDDDR (CDADR X))) )
      (MAPCAR
         '(LAMBDA (Y)
            (PRINTC (CONCAT '"       " (CAR Y) '" "
                    (CADR Y) '. (CADDR Y) (CAR (CDDDR Y)) ))
            (PRIN (CADR (CDDDR Y)) ))
         (CADDR X) )
      (PRINTC " ")
       T ))
```

FICHIER FORM2.LSP :

```
(DEFUN FORMATE
   (LAMBDA (X)
       (SETQ *ID1 (OUTPUT '"GENERE.DOC"))
       (PRINTC (CONCAT (CAAR X) '" (" (CADAR X) '=
               (CAR (CDDAR X)) '. (CADR (CDDAR X)) ') ) *ID1 )
       (PRINTC (CONCAT (CAADR X) '" " (CAR (CDADR X)) '.
               (CADR (CDADR X)) (CADDR (CDADR X)) ) *ID1 )
       (PRIN (CAR (CDDDR (CDADR X))) *ID1 )
       (MAPCAR
         '(LAMBDA (Y)
            (PRINTC (CONCAT '"       " (CAR Y) '" " (CADR Y) '.
                    (CADDR Y) (CAR (CDDDR Y)) ) *ID1)
            (PRIN (CADR (CDDDR Y)) *ID1 ))
         (CADDR X) )
      (PRINTC " " *ID1 )
      (CLOSE *ID1)
       T ))
```

FICHIER GENERE.LSP :

```
(DEFUN GENERE
  (LAMBDA ( OBJ PAR )
    (FORMATE (LIST
               (SUBST 'RES (CADR OBJ)
                  (SUBST 'TOBJ (CAR OBJ)
                         '(RETRIEVE RES TOBJ CODE) ))
               (SUBST 'TOBJ (CAR OBJ)
                  (SUBST 'PROP (CAAR PAR)
                     (SUBST 'OP1 (CADAR PAR)
                        (SUBST 'VAL (CAR (CDDAR PAR))
                               '(WHERE TOBJ PROP OP1 VAL) ))))
               (MAPCAR
                 '(LAMBDA (ELEM)
                   (SUBST 'OP2 (CAR ELEM)
                     (SUBST 'TOBJ (CAR OBJ)
                       (SUBST 'PROP (CADR ELEM)
                         (SUBST 'OP1 (CADDR ELEM)
                           (SUBST 'VAL (CAR (CDDDR ELEM))
                                  '(OP2 TOBJ PROP OP1 VAL) ))))))
                  (CADR PAR) )))))
```